# XML-dokumenttien deklaratiivinen kyselykieli

Teemu Kumpulainen

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyopin laitos
Teemu Kumpulainen: XML-dokumenttien deklaratiivinen kyselykieli
Pro gradu -tutkielma, 73 sivua, 24 liitesivua
Toukokuu 2003

---

Tutkielma käsittelee XML:llä esitettyihin rakenteisiin dokumentteihin kohdistuvaan tiedonhakuun soveltuvia kyselykieliä. Tutkielmassa tarkastellaan piirteitä, jotka tällaisen kielen olisi sisällettävä sekä ilmaisuvoiman että käytettävyyden näkökulmasta. W3-konsortiumi kehittää XQuery-kyselykieltä ja sitä tultaneen suosittelemaan tulevaisuudessa XML-kyselykielen standardiksi. Kuitenkin XQueryssä on joitakin ongelmallisia piirteitä. Esimerkiksi soveltaessaan XQueryn primitiivejä käyttäjä joutuu iteratiivisten ilmaisuiden avulla eksplisiittisesti määrittelemään tulosdokumentin elementtien keskinäiset suhteet. Tutkielmassa annetaan kyselyesimerkkejä, jotka osoittavat, että määrittelyt XQuerylla muistuttavat enemmän algoritmin suunnittelua kuin suoraviivaista deklaratiivista spesifiointia. Tutkielmassa kehitetään rakenteisten dokumenttien käsittelyyn kyselykieli, jonka deklaratiivisuuden aste on huomattavasti korkeampi kuin Xqueryn. Kieli koostuu kolmesta selkeästi toisistaan erottuvasta osasta. Kielen tulososassa kuvataan tulosdokumentin rakenne. Määrittelyosassa ilmaistaan kyselyssä tarvittavat muuttujat, jotka ovat analogisia logiikkaohjelmoinnin muuttujakäsitteelle. Muuttuja viittaa aina johonkin rakenteiseen dokumenttiin tai sen osaan. Ehto-osassa muuttujien avulla määritellään lähtödokumentin/lähtödokumenttien elementtien ja tulosdokumentin elementtien väliset suhteet. Tutkielman kieli toteutettiin DCG-logiikkakieliopin avulla. Tutkielmassa vertaillaan kehitetyn kyselykielen ja XQueryn määrittelyjen eroavaisuutta useiden esimerkkikyselyjen yhteydessä.

Avainsanat ja -sanonnat: XML, XQuery, kyselykielet, DCG (Definite Clause Grammar)

## Sisällysluettelo

# 1. Johdanto

Tutkielma tarkastelee rakenteisten dokumenttien kuvauskielellä XML:lla esitettyihin dokumentteihin kohdistuvaan tiedonhakuun soveltuvia kyselykieliä ja esittää näkemyksen siitä, millainen tällaisen kielen olisi oltava. Aluksi tarkastellaan jo olemassa olevia tai toteutusvaiheessa olevia kieliä, erityisesti W3-konsortiumin työstämää XQuery-kieltä. Siihen liittyvien esimerkkien yhteydessä pyritään osoittamaan XQueryn ilmaisuihin liittyviä käytettävyysongelmia. Vaikka siis on jo olemassa esitys XML-kyselykielen standardista, se sisältää sellaisia piirteitä, että tutkielman seuraavissa luvuissa esiteltävän, XQueryn lähestymistavasta poikkeavan, rakenteisten dokumenttien kyselykieli on tarpeen.

Aluksi käsitellään XML-kielen yleisiä piirteitä, siinä määrin kuin tutkielman yhteydessä tarvitaan. Lisäksi esitellään XPath- ja XSLT-kieliä, perehdytään XQueryyn, ja katsotaan muutaman esimerkin avulla, kuinka XQuery ja XSLT eroavat toisistaan ilmaisuvoimaltaan. Tämän jälkeen tarkastellaan ongelmia, joita XQueryn ominaispiirteisiin liittyy ja todetaan, että on olemassa tarve XQueryä deklaratiivisemmalle kyselykielelle, joka esitetään tutkielmassa.

## 1.1. Yleisiä XML-kielen käsitteitä

Tässä kohdassa annetaan yleiskuva XML-kielen käsitteistä ja tutustutaan tietyssä määrin XML-muotoisen tiedon käsittelyssä käytettäviin työkaluihin.

### 1.1.1. XML-kielen perusteet

XML (eXtended Markup Language) on rakenteisen tekstin merkintäkieli, joka on tarkoitettu ennenkaikkea työkaluksi esittää dokumentteja tietoverkossa. Se tarjoaa HTML-kieltä monipuolisemmat mahdollisuudet tiedon prosessointiin. W3-konsortiumi julkaisi vuonna 1998 XML-kielen virallisen suosituksen. Nyt on käytössä vuonna 2000 revisoitu versio. [XML]

XML-muotoisen tiedon perusrakenteena pidetään elementtiä, joka muistuttaa pienin eroin muiden merkintäkielten vastaavaa rakennetta. Elementti koostuu alkutunnisteesta (start tag), sisällöstä ja lopputunnisteesta (end tag). Sisältönä voi olla merkitsemätöntä tekstiä tai toisia elementtejä. Alkutunniste koostuu merkin aloittavasta pienempi kuin -merkistä (<), elementin nimestä, mahdollisista välilyönnein erotetuista attribuuteista ja merkin lopettavasta suurempi kuin -merkistä (>). Attribuutit ovat eräänlaisia avain-arvopareja, joissa annetaan attribuutin nimi, yhtäsuuruusmerkki (=) ja attribuutin arvo lainausmerkkien (") sisään kirjoitettuna. Niitä voidaan käyttää ilmaisemaan informaatiota, joka liittyy elementin varsinaiseen sisältöön, mutta jota ei

kuitenkaan katsota olennaiseksi sisällöksi. Lopputunnisteen nimi on sama kuin alkutunnisteenkin, mutta siinä on aloittavan pienempi kuin-merkin jälkeen kauttaviiva ( / ) eikä se voi sisältää attribuutteja. Esimerkiksi

```
<entry day="Monday">The morning was chilly and dull.</entry>
```

on XML-muotoinen elementti. (XML-koodi on luettavuuden vuoksi kirjoitettu tässä ja muissa esimerkeissä tasalevyisellä kirjasintyylillä.) Elementin, joka alkaa jonkun toisen elementin sisällä, on myös päätyttävä siellä. Tämän vuoksi esimerkiksi

```
<a> <b> Text. </a> </b>
```

ei kelpaa XML-merkinnäksi. Elementti b alkaa a:n sisällä, mutta b:n lopputunniste on a-elementin ulkopuolella. Edelliset tunnisteet voitaisiin järjestää XML-merkinnäksi joko

```
<a> <b> Text. </b> </a>
```

tai

```
<b> <a> Text. </a> </b>.
```

Erikoismerkintänä tyhjä elementti voidaan lyhentää jättämällä lopputunniste pois ja merkitsemällä kauttaviiva ennen alkutunnisteen päättävää suurempi kuin -merkkiä. Tällöin elementit

```
<entry day="Monday"></entry>
```

ja

```
<entry day="Monday"/>
```

tarkoittavat samaa asiaa.

### 1.1.2. Dokumentin kieliopin määrittely

Hyvin muodostettu XML-dokumentti sisältää mahdollisten kommenttien lisäksi johdanto-osan (prolog) ja yhden juurielementin (root), jonka sisälle kaikki muut elementit sijoittuvat. Johdanto-osassa voidaan antaa muunmuassa kyseisen dokumentin rakenteeseen liittyvää informaatiota, kuten dokumentin kieliopin määrittely.

Erona perinteisesti dokumenttien esittämisessä käytettyyn HTML-kieleen nähden XML sellaisenaan ei tarjoa kiinteää tietyllä tavalla nimettyä elementtijoukkoa, vaan elementit voidaan nimetä periaatteessa vapaasti. XML-muotoinen, tiettyyn sovellusalueeseen liittyvä tieto ilmaistaan yleensä tämän sovellus–alueen puitteissa yhtenäistetyssä muodossa, joka määrätään tarkoitusta varten tehdyllä DTD-määrittelyllä (Document Type Definition). DTD-määrittelyn avulla voidaan siis antaa kielioppi tietyn XML-sovelluksen dokumenteille, mutta se on melko suppea ja joissakin yhteyksissä on tarve tarkemmalle tasolle yltävälle määrittelylle. XML-sovelluksena toteutettu XML

Schema tarjoaa DTD-notaatiota monipuolisemmat mahdollisuudet kieliopin määrittämiseen. [XMLSchema] Tässä sitä ei kuitenkaan käsitellä. DTD esitellään tarkemmin esimerkkidokumenttien yhteydessä.

XML-kielen vahvuus on tiedon kuvaamisessa. Sen avulla voidaan esittää hyvin monenlaisia asioita, joihin perinteinen HTML ei kykene. Koska XML:n merkkaus perustuu nimenomaan tietoon eikä ulkoasuun, on se sovelluksineen jäsentyneessä tiedonhaussa erittäin mielenkiintoinen tutkimuksen kohde.

### 1.2. XPath ja XSLT tiedonhaussa

Tässä luvussa käsitellään XML-tiedon käsittelemiseen tarvittavia työkaluja. Näitä ovat XML-elementteihin viittaamiseen tarkoitettu XPath, dokumenttien uudelleenmuotoiluun kykenevä XSLT-kieli ja pelkästään kyselykieleksi suunniteltu XQuery-kieli. XSLT- ja XQuery-kielten, esittelyn jälkeen tarkastellaan niiden välisiä eroja.

#### 1.2.1. XPath

Xpath-kielellä voidaan osoittaa johonkin/joihinkin XML-dokumentin osaan/osiin [XPath, 1999]. XPath näkee XML-dokumentin puurakenteena, jonka solmuina on XML-elementtejä. Elementteihin voidaan viitata erityisillä polkumäärityksillä, joihin saatetaan liittää identifioivia predikaatteja.

```
<diary author="Wally Week">
  <entry day="Monday">The morning was chilly and dull.</entry>
  <entry day="Tuesday">The rain was falling all the day.</entry>
  <entry day="Wednesday">The fog was quite thick.</entry>
  <entry day="Thursday">The sunset was beautiful.</entry>
  <entry day="Friday">The night was clear.</entry>
  <entry day="Saturday">The headache was devastating.</entry>
  <entry day="Sunday">The sun was shining.</entry>
</diary>
```

Kuva 1.1: XML-dokumentti.

Esimerkiksi kuvan 1 dokumentissa voidaan viitata mihin tahansa `entry`-nimiseen elementtiin XPath-ilmaisulla `/diary/entry` tai pelkästään `entry`. Tämä edellyttää, että aiemmin on ilmaistu, että ollaan käsittelemässä Dokumentti 1:tä. Haluttaessa viitata maanantaihin (attribuutin `day` arvoa `monday`) liittyvään `entry`-elementtiin, ja tiedämme, että maanantai on ensimmäinen `entry`, voidaan antaa ilmaisu

`//entry[1]` (XPath-lauseke 1a),

koska kyseinen elementti on ensimmäisenä dokumentissa 1, tai ilman tietoa elementin sijainnista vaikkapa

`/diary/entry[@day='Monday']` (XPath-lauseke 2a).

Viittaus voi olla absoluuttinen, jolloin määritellään koko polku juurielementistä lähtien, tai suhteellinen. Hakasulkeiden välissä annettava lauseke voi olla myös jokin XPathin funktioista, joilla voidaan käsitellä mm. elementin tekstisisältöä ja laskea aritmeettisia lausekkeita. XPathin syntaksissa käytetään erinäköisiä lyhenteitä joista muutamia itse asiassa käytettiin jo, kun annettiin lauseke 1a, jonka hakasulkeiden välinen osa kokonaisuudessaan kirjoitettuna olisi `position() = 1`. Mikäli XPathin kauttaviivamerkinnän (`/`) jälkeen ei ole kerrottu erikseen askeltyyppiä, on kyse `child`-tyyppisestä polkuaskeleesta. Kaksi viivaa (`//`) on puolestaan lyhenne `descendant-or-self`-tyyppisestä polkuaskeleesta. Kokonaisuudessaan lyhentämätön ilmaisu lausekkeelle 1a olisi

```
/descendant-or-self::node()/child::entry[position()=1]
```

(XPath-lauseke 1b)

ja lausekkeen 2a lyhentämätön versio olisi

```
/child::diary/child::entry[attribute::day=1]
```

(XPath-lauseke 2b).

Lausekkeessa 2b käytetään askeltyyppiä `attribute`, joka voidaan lyhentää @-merkillä. Lisäksi yleinen askeltyyppi on `parent` (vanhempisolmu), joka voidaan lyhentää kahden pisteen merkinnällä (..). Esimerkiksi diary-elementtiin voitaisiin viitata `entry`-elementistä merkinnällä

`entry/..` (lyhentämättömänä `entry/parent::node()` ).

XPathin ilmaisut ovat olennainen osa XSLT- ja XQuery-kieliä. Tällä hetkellä W3-konsortiumin suosittelema versio on 1.0, mutta kehitteillä on uusi versio 2.0, jota XQueryn on tarkoitus käyttää [Xpath, 2002].

### 1.2.2. XSLT

XSLT (Extensible Stylesheet Language Transformation) -kieli on XML-dokumenttien muuntamiseen tarkoitettu kieli [XSLT, 1999]. Sillä voidaan muuntaa XML-dokumentti esimerkiksi XHTML-dokumentiksi (HTML-kieli ilmaistuna XML-notaatiolla), mutta sillä voidaan myös rakentaa täysin uudenlaisia XML-elementtejä, tai valita vain tietty joukko esitettävään muotoon esittämällä elementtien sisältöihin tiettyjä ehtoja. Tällä hetkellä XSLT:n versio 1.0 on ainoa yleisesti käytössä oleva XML-kyselykieli.

XSLT on XML-sovellus, eli XSLT-dokumentti koostuu XML-elementeistä. Periaattena on, että muokkauksen lähtödokumentin tai -dokumenttien elementtejä valitaan XPath-lausekkeilla ja tuotetaan tulokseksi XSLT-

dokumentin määrittämää tekstiä. XSLT:stä on suunnitteilla myös versio 2.0, joka XQueryn tapaan tukisi XPath versiota 2.0 [XSLT, 2002].

### 1.2.3. Yksinkertainen XSLT-esimerkki

Esimerkit on tuotettu käyttäen Microsoft Windows -käyttöjärjestelmille tehtyä MSXSL-ohjelmaa, joka tuottaa XML- ja XSL-dokumenttien perusteella uuden dokumentin [MSXSL].

```
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml"/>
<xsl:template match="diary">
<content>
Diary of <xsl:value-of select="@author"/>.
----------------------------------------
<xsl:apply-templates/>
----------------------------------------
</content>
  </xsl:template>
   <xsl:template match="entry"><xsl:value-of select="@day"/>:
     <xsl:apply-templates/>
  </xsl:template>
</xsl:stylesheet>
```

Kuva 1.2. XSLT-esimerkki 1.

Jos lähtödokumentti on aiemmin esitetty dokumentti, ja meillä on kuvan 1.2 XLST-ilmaisu, niin tulokseksi saadaan kuvan 1.3 dokumentti.

```
<content>
Diary of Wally Week.
----------------------------------------
-
Monday: The morning was chilly and dull.
Tuesday: The rain was falling all the day
Wednesday: The fog was quite thick.
Thursday: The sunset was beautiful.
Friday: The night was clear.
Saturday: The headache was devastating.
Sunday: The sun was shining.
----------------------------------------
```

Kuva 1.3. XSLT-ilmaisun tuottama tulosdokumentti.

XSLT-esimerkissä 1 määrätään `xsl:output`-elementin `method`-attribuutilla tulosdokumentin olevan XML:ää. Esimerkissä on tehty lähtödokumentin elementtejä varten valmiita malleja (template), jotka toimivat siten, että `xsl:template`-elementtien attribuutti `match` saa arvokseen XPath-ilmaisuja, joiden osoittamat elementit käsitellään `xsl::template`-elementin sisällä. XPath-lausekkeet, joita `match`-attribuutti saa sisältää, koostuvat vain `attribute` ( /@ )- tai `child` ( / ) -tyyppisistä polkuaskelista. Mahdollisten alielementtien sijainti

tulosdokumentissa määrätään `xsl::apply-templates`-elementillä. Mikäli jollekin alielementille ei löydy mallia, tulostuu se muokkaamattomana tulosdokumenttiin, olettaen, että XSLT-dokumentissa on käytetty `xsl:apply-templates`-elementtiä.

Toinen tapa siirtää tietoa lähtödokumentista tulosdokumenttiin, on käyttää `xsl:value-of` -elementtiä, jonka `select`-attribuutin arvo on jälleen XPath-polkumääritys. Tällöin polun osoittama elementti sijoittuu elementin osoittamaan paikkaan. Esimerkissä tätä käytettiin viikonpäivien tulostamiseen.

XSLT tarjoaa lisäksi muita ohjausrakenteita, joiden avulla XSLT-kieltä voidaan itse asiassa käyttää vaikkapa rekursiiviseen ohjelmointiin, kuten esimerkiksi Nykänen [2001] kirjassaan esittää. Tällainen soveltaminen on kuitenkin työlästä, sillä tämä vaatii XSLT:n tulosdokumentin muunnosprosessin ohjailua ohjausrakenne-elementtien avulla. Yleensä on löydettävissä mielekkäämpiä menetelmiä. Yleisesti XSLT-kielen ongelma kyselykielenä on sen painottuminen rakenteellisen muotoilun ilmaisemiseen. Tietotarpeita varten XSLT:ssä käytetään XPath-ilmaisuja, jotka jäävät hyvin suppeiksi osasiksi koko XSLT-ilmaisua. XSL-dokumentista tulee helposti monimutkainen ja sen suunnitteluun ja tekemiseen voi kulua kohtuuttoman pitkiä aikoja. XSLT tarjoaa mahdollisuudet XML-tiedon monipuoliseen prosessointiin mutta myös vaatii käyttäjältään paljon. XSLT- ilmaisussa käytetään samalla notaatiolla tuotettuja elementtejä sekä ilmaisemaan prosessointia että prosessoinnin tulosta. Suuremmissa kokonaisuuksissa on kyettävä erottamaan, mitkä elementit kuuluvat XSLT-ilmaisuun ja mitkä tulokseen. Käyttäjän on kyettävä lukemaan rakenteisia dokumenteja siten, että hän ymmärtää notaation tuottaman rakenteen. Lisäksi XSLT-ilmaisussa käytetään ohjausrakenteita, joita yleensä käytetään proseduraalisissa ohjelmointikielissä. Näin ollen ilman ohjelmointitaitoja ja kykyä algoritmiseen ajatteluun XSLT-ilmaisun konstruointi on liian monimutkainen toimenpide.

### 1.3. XQuery

XQuery on kehitteillä, koska on katsottu tarvittavan jatkuvasti kasvavan XML-muotoisen tiedon älykkääseen kyselyyn kykenevää kieltä [XQuery, 2002]. Vaikka XSLT on ilmaisuvoimaltaan XQueryn tasoa [Lechner et al., 2002], on silti katsottu, että on tarve helppokäyttöisemmälle kielelle.

Selkein ero XSLT-kieleen on se, että XQuery ei ole toteutettu XML-muotoisesti. XQuery 1.0 on suunnitteilla oleva laajennos kehiteltävään XPathin 2.0-versioon. Versio 2.0 laajentaa XPath 1.0-version funktio- ja operaatiojoukkoa ja lisää mm. tietotyyppejä päivämääriä ja muita vastaavia tietoja varten. XQueryn 1.0- ja XPathin 2.0-version operaatioita koskevassa, 15.10.2002

julkistetussa luonnoksessa esitellään yhteensä 251 eri funktiota tai operaattoria, jotka aiotaan sisällyttää kieleen [XQuery op, 2002]. Voitaneen todeta, että tässä mielessä käyttäjältä vaadittavat edellytykset XQueryn hallintaan ovat sitä luokkaa, mitä yleensä rajapintojen avulla ohjelmoitaessa tarvitaan. Tämä tekee väitteen XQueryn helppokäyttöisyydestä varsin kyseenalaiseksi.

### 1.3.1. FLWOR-lauseke

XPath 2.0 -lausekkeiden lisäksi XQuery tarjoaa niinsanotun FLWOR-lauserakenteen (For-Let-Where-Order-Return), jonka avulla se tukee muuttujien iterointia ja sitomista väliaikaisiin tuloksiin. FLWOR-lause koostuu vähintään for- tai let- ja return-lauseesta. For- ja let-lauseet määrittävät muuttujat XPath-lausekkeilla ja return-lause tuottaa FLWOR-lauserakenteen tulosteen. Where- ja order-lauseet testaavat ja rajoittavat muuttujien sisältöä ja uudelleenjärjestävät tulosta.

For- ja let- lauseet eroavat siinä, että let-lauseen muuttuja arvotetaan uudelleen jokaista for-lauseen iteraatiokierrosta kohti. Return-lauseen palauttama teksti tulostuu samaten kerran jokaista for-lausetta kohti. Jos for-lausetta ei ole, suoritetaan let-lauseen arvotus kuten tuloksessa olisi yksi (tyhjä) for-lause mukana. Kyseessä on tyypillinen kaksoissilmukka-ajattelu. Seuraavat esimerkit XQueryn spesifikaatioluonnoksesta [XQuery, 2002] valaisevat asiaa. Kysely

```
let $s := (<one/>, <two/>, <three/>)
return <out>{$s}</out>
```

tuottaa tulokseksi out-elementin

```
<out>
   <one/>
   <two/>
   <three/>
</out> ,
```

kun taas for-lausetta käytettäessä kysely

```
for $s in (<one/>, <two/>, <three/>)
return <out>{$s}</out>
```

tuottaa tulokseksi elementit

```
<out>
   <one/>
</out>
<out>
   <two/>
</out>
<out>
   <three/>
</out> .
```

Huomataan, että käytettäessä let-lausetta saadaan yksi out-elementti, jonka sisään kaikki muuttujan arvotukset tulevat, ja for-lauseessa jokaista arvotusta kohti saadaan oma out-elementti. Usein FLWOR-lauseesta käytettään myös nimitystä FLWR. Lause lisää XPathin deklaratiiviseen tapaan esittää asiat proseduraalisista ohjelmointikielistä tutun silmukkarakenteen. XQueryn kyselyt rakennetaan tämän rakenteen luomaan kehikkoon.

### 1.3.2. XQuery ja XSLT –esimerkkejä ja vertailuja

Yleiskuvan saamiseksi tässä esitetään yksinkertainen esimerkki kielestä. Se on otettu XQueryn käyttötapakuvauksesta [XQuery use, 2002]. Kyselyssä käytetään tiedonhaun kohteena XML-dokumenttia, jonka URL-osoitteeksi oletetaan http://www.bn.com/bib.xml (kuva 1.4).

```
<bib>
    <book year="1994">
        <title>TCP/IP Illustrated</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price> 65.95</price>
    </book>
    <book year="1992">
        <title>Advanced Programming in the Unix environment</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
    </book>
    <book year="2000">
        <title>Data on the Web</title>
        <author><last>Abiteboul</last><first>Serge</first></author>
        <author><last>Buneman</last><first>Peter</first></author>
        <author><last>Suciu</last><first>Dan</first></author>
        <publisher>Morgan Kaufmann Publishers</publisher>
        <price>39.95</price>
    </book>
    <book year="1999">
        <title>The Economics of Technology and Content for Digital
         TV</title>
        <editor>
               <last>Gerbarg</last><first>Darcy</first>
                <affiliation>CITI</affiliation>
        </editor>
            <publisher>Kluwer Academic Publishers</publisher>
        <price>129.95</price>
    </book>
</bib>
```

Kuva 1.4. Dokumentti bib.xml.

Dokumentissa on siis kuvattu book-elementtien avulla joitakin kirjoja, joiden nimi-, tekijä- ja muita vastaavia tietoja kuvaamaan on käytetty elementtejä `title`, `author` ja niin edelleen. Nyt halutaan tietää niiden bib.xml-

dokumentissa kerrottujen kirjojen julkaisuvuosi ja nimeke, jotka on julkaissut Addison-Wesley vuoden 1991 jälkeen. Kysely annetaan kuvassa 1.5.

```
<bib>
          {
          for $b in document("http://www.bn.com/bib.xml")/bib/book
          where $b/publisher = "Addison-Wesley" and $b/@year > 1991
          return
          <book year="{ $b/@year }">
                    { $b/title }
          </book>
          }
</bib>
```

Kuva 1.5: XQuery-kysely 1.

Tulokseksi saadaan kuvan 1.6 XML-elementti.

```
<bib>
 <book year="1994">
   <title>TCP/IP Illustrated</title>
 </book>
 <book year="1992">
   <title>Advanced Programming in the Unix
environment</title>
 </book>
</bib>
```

Kuva 1.6: XQuery-esimerkin 1 tulos.

Kyselyssä määritellään for-sanalla alkavalla rivillä Xpath-lausekkeen (`document("http//..")/bib/book` ) avulla muuttuja `$b`, joka osoittaa dokumentin `book`-elementteihin. Aaltosuluilla ( { , }) merkitty ulompi lohko suoritetaan järjestyksessä rivi kerrallaan niin monta kertaa kuin `book`-elementti löytyy. Jokaisella iteraatiokierroksella where-osassa tarkistetaan, täyttääkö muuttujan osoittama elementti annetut ehdot. Tulosteen tuottavassa return-osassa annetaan XML-muotoista tekstiä, johon on upotettu aaltosulkujen merkitsemiin lohkoihin XPath-lausekkeita, jotka osoittavat johonkin elementtiin.

XQuery-esimerkkikysely 1 on melko suoraviivaisesti käännettävissä XSLT:ksi. Vastaavan tuloksen tuottava XSLT-dokumentti olisi kuvan 1.7 mukainen.

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <bib>
    <xsl:for-each
        select="bib/book[publisher='Addison-Wesley' and @year>1991]">
    <xsl:variable name="year" select="@year"/>
         <book year= "{$year}" ><xsl:copy-of
select="./title"/></book>
   </xsl:for-each>
</bib>
</xsl:template>
</xsl:stylesheet>
```

Kuva 1.7. XSLT-esimerkki 2.

XSLT-esitys vaatii hieman enemmän lukijaltaan, mutta voidaan huomata, että varsinainen kyselyssä käytetty semantiikka saadaan miltei identtisellä XPath-lausekkeella (merkitty XSLT-esimerkkiin 2 paksunnetulla kirjasimella) `publisher='Addison-Wesley' and @year>1991` , erona on vain se, että XQueryssa lauseke on annettu where-osassa muuttujan avulla viittaamalla ja XSLT-versiossa viitattavan elementtisolmun `bib/book` predikaattina.

Seuraavassa, hieman monimutkaisemmassa kyselyssä, XQueryssä joudutaan käyttämään sisäkkäisiä ilmaisuja. Jokaista yksittäistä kirjoittajaa (`author`-elementtiä) kohti haetaan tämän kirjoittamien teosten nimet. XQuery ei tarjoa mainitun kaltaisen ongelman ratkaisevaa yhteen kyselylausekkeeseen perustuvaa ilmaisua, vaan ratkaisu on suunniteltava proseduraalisten ohjelmointikielten tapaan kahden sisäkkäisen silmukkarakenteen eli FLWR-lauseen avulla, kuten on esitetty kuvassa 1.8.

```
<results>
  { for $a in distinct-values(
                  document("http://www.bn.com/bib.xml")//author)
    return
      <result>
        { $a }
        { for $b in document("http://www.bn.com/bib.xml")/bib/book
          where some $ba in $b/author satisfies deep-equal($ba,$a)
          return $b/title
        }
      </result>
  }
</results>
```

Kuva 1.8: XQuery-kysely 2.

Koska dokumentissa bib.xml voi esiintyä sama `author`-elementti usemmassa `book`-elementissä, käytetään XPath 2.0 `distinct-values` -funktiota eliminoimaan samansisältöiset `author`-elementit. Ulompi kysely tulostaa haetut elementit ja sisemmän kyselyn, joka suoritetaan uudelleen jokaiselle muuttujan

$a arvolle (yksi iteraatiokierros). Where-osan avainsana some vaatii, että vain jonkun muuttujan $ba esittämistä elementeistä on täytettävä vaadittu ehto. Funktio deep-equal tutkii, että sen saamien parametrien sisältö ja elementtisolmujen nimet ovat samat. Xquery-kysely 2 tuottaa tulokseksi kuvassa 1.9 annetun XML-elementin.

```
<results>
  <result>
    <author><last>Stevens</last><first>W.</first></author>
    <title>TCP/IP Illustrated</title>
    <title>Advanced Programming in the Unix
environment</title>
  </result>
  <result>

<author><last>Abiteboul</last><first>Serge</first></author>
    <title>Data on the Web</title>
  </result>
  <result>
    <author><last>Buneman</last><first>Peter</first></author>
    <title>Data on the Web</title>
  </result>
  <result>
    <author><last>Suciu</last><first>Dan</first></author>
    <title>Data on the Web</title>
  </result>
</results>
```

Kuva 1.9: XQuery-kyselyn 2 tulos.

Tuloksen result-elementtien sisältönä on yksi author-elementti, ja siihen liittyvät title-elementit.

Seuraavaksi annamme kyselylle XSLT-kielisen määrittelyn. Koska XSLT-kielen nykyisin suositeltu versio 1.0 ei tunne XPath-versiota 2.0, kierrämme disctinct-values-funktion puutteen käyttämällä XPathin askelpolkutyyppiä preceding vertaamaan kulloisella xsl:for-each-elementin iterointikierroksella haettua arvoa dokumentissa edeltäneisiin arvoihin. Kuvan 1.10 XSLT-ilmaus tuottaa tulokseksi saman vastauksen kuin XQuery-kysely 2. XQuery-kyselyn kahta sisäkkäistä FLWR-lausetta vastaavat XSLT:ssä myöskin sisäkkäiset xsl:for-each-elementit, joista sisemmässä on hyödynnetty muuttujaa, joka on sidottu ulomman xsl:for-each-elementin kontekstisolmuun. Tässäkin kyselyssä erona on lähinnä XSLT-kielen työläs notaatio.

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
<results>
    <xsl:for-each select="//author[not(. = preceding::*//author)]">
  <result>
      <xsl:variable name="a" select="."/>
      <xsl:copy-of select="$a"/>
      <xsl:for-each select="//book[author=$a]">
        <xsl:copy-of select="./title"/>
      </xsl:for-each>
  </result>
     </xsl:for-each>
</results>
    </xsl:template>
  </xsl:stylesheet> >
```

Kuva 1.10. XSLT-esimerkki 3.

Rakenteellisesti `xsl:for-each`-ilmaus tarjoaa samat ilmaisumahdollisuudet kuin XQueryn FLWR-lauserakenne. Ilmaisuvoimassa ero syntyykin lähinnä XPathin versioeroista, eli kun XSLT alkaa tukea XPath versiota 2.0[1], ilmaisuvoiman puolesta voidaan yhtä hyvin käyttää XSLT:tä kuin XQueryä.

### 1.3.3. XSLT:n ja XQueryn erot ja ongelmat

Kuten edellä annetut esimerkit osoittavat, XSLT ja XQuery pystyvät ilmaisemaan samat asiat. Koska XQuery on suunniteltu XPath 2.0-versiolle, se tarjoaa kuitenkin laajemmat funktiokirjastot ja tyyppimäärittelyt, kuin mihin XSLT kykenee. Kuitenkaan XSLT:n versio 2.0 ei ole enää tässäkään suhteessa heikompi, sillä se on suunniteltu samaiselle XPathin versiolle kuin XQuerykin.

Ero syntyy lähinnä käytettyjen merkintöjen tuomista piirteistä. Koska XSLT tukeutuu XML-notaatioon, on sitä tietyissä tapauksissa hankalaa tuottaa. Toisaalta tämä antaa XSLT-syntaksille selkeän rakenteen, joka on loogisesti seurattavissa. Lisäksi XSLT on jo laajalti käytössä ja sitä tuetaan hyvin eri ympäristöissä. Sitä ei ole suunniteltu eksplisiittisesti tiedonhakua varten, vaikka se käy myös tähän tarkoitukseen.

XQuery tarjoaa oman merkintätapansa, joka ei kuitenkaan ole täysin ongelmaton sekään. XQuery on käännettävissä keskeisiltä osiltaan lähes suoraan XSLT:ksi [Lechner et al., 2002; Lenz, 2002]. XQuery-kielestä on pyritty luomaan helppokäyttöinen ja helposti luettava, mutta se ei aina ole sitä. Osittain helppokäyttöisyyttä nakertaa XPath 2.0-version funktioiden ja operaatioiden suuri määrä; voi olla vaikea löytää kohtuullisen ajan sisällä juuri

[1] Kuten aiemmin on mainittu, W3-konsortiumi kehittelee XSLT:stä versiota 2.0, joka käyttää XPath 2.0-ilmaisuja.[XSLT, 2002]

johonkin tiettyyn kyselyyn sopiva funktio satojen joukosta. Lisäksi XQueryn tehokas käyttö vaatii käyttäjäänsä ymmärtämään suhteellisen syvällisesti, kuinka kielen ilmaisun prosessointi tapahtuu proseduraalisesti, mikä ei välttämättä tarjoa kyselyn tekijän näkökulmasta vaivattominta lähestymistapaa, jossa kyselykielen käyttäjän tulisi voida keskittyä kyselyn toteutuksen sijasta sen merkityksen ilmaisemiseen.

XQueryn suurimpia ongelmia on se, että XQuerylla kysely on ilmaistava samoin periaattein tai samanlaisen ajatuksellisen prosessin tuloksena kuin XSLT-kielessäkin, eli perinteisen proseduraalisen ohjelmointikielen periaattein. Kyselyssä on otettava huomioon, miten ja missä järjestyksessä kyselyn muuttujien arvotukset tapahtuvat ja niiden keskinäiset suhteet. Lisäksi käyttäjän on ymmärrettävä kyselyn suorituksen proseduraalinen kulku, mikä vie huomion pois olennaisesta eli tietotarpeiden esittämisestä. Seuraava esimerkki pyrkii valaisemaan asiaa. Se on lyhennetty versio XQueryn käyttötapakuvauksesta [XQuery use, 2002] poimitusta tekstihakua esittelevästä kyselystä.

Oletetaan, että on olemassa XML-dokumentit "news.xml" ja "company-data.xml" (kuvat 1.11 ja 1.12).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<company>
  <name>Foobar Corporation</name>
  <partners>
    <partner>YouNameItWeIntegrateIt.com</partner>
    <partner>TheAppCompany Inc.</partner>
  </partners>
  <competitors>
    <competitor>Gorilla Corporation</competitor>
  </competitors>
</company>
```

Kuva 1.11. Dokumentti company-data.xml.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<news>
<news_item>
   <title> Gorilla Corporation acquires YouNameItWeIntegrateIt.com
</title>
   <content>
      <par> Today, Gorilla Corporation announced that it will purchase
          YouNameItWeIntegrateIt.com. The shares of
          YouNameItWeIntegrateIt.com dropped $3.00 as a result of this
          announcement.
      </par>
      <par> As a result of this acquisition, the CEO of
          YouNameItWeIntegrateIt.com Bill Smarts resigned. He did not
          announce what he will do next.  Sources close to
          YouNameItWeIntegrateIt.com hint that Bill Smarts might be
          taking a position in Foobar Corporation.
      </par>

      <par>YouNameItWeIntegrateIt.com is a leading systems integrator
          that enables <quote>brick and mortar</quote> companies to
          have a presence on the web.
      </par>
   </content>
   <date>1-20-2000</date>
   <author>Mark Davis</author>
   <news_agent>News Online</news_agent>
</news_item>
<news_item>
   <title>Foobar Corporation releases its new line of Foo products
   today</title>
   <content>
      <par> Foobar Corporation releases the 20.9 version of its Foo
            products.  The new version of Foo products solve known
            performance problems which existed in 20.8 line and
            increases the speed of Foo based products tenfold. It also
            allows wireless clients to be connected to the Foobar
            servers.
      </par>
      <par> The President of Foobar Corporation announced that they
            were proud to release 20.9 version of Foo products and
            they will upgrade existing customers <footnote>where
            service agreements exist</footnote>
            promptly. TheAppCompany Inc. immediately announced that it
            will release the new version of its products to utilize
            the 20.9 architecture within the next three months.
      </par>
      <figure>
          <title>Presidents of Foobar Corporation and TheAppCompany
          Inc. Shake Hands</title> <image source="handshake.jpg"/>
      </figure>
   </content>
   <date>1-20-2000</date>
   <news_agent>Foobar Corporation</news_agent>
</news_item>
</news>
```

Kuva 1.12. Dokumentti news.xml.

Kyseisistä dokumenteista halutaan saada selville, missä uutisjutuissa on mainittu yritys nimeltä Foobar Corporation ja joku sen yhteistyökumppaneista (`partner`) samassa jutussa (`news_item`), ja että juttu ei ole yrityksen itsensä julkaisema (`news_agent`). Tiedetään, että yhteistyökumppaneiden nimet on annettu company-data.xml-dokumentin `partner`-elementeissä. XQuery-ilmaisu on annettu kuvassa 1.13.

```
define function partners($company as xs:string) as element* {
  let $c := document("company-data.xml")//company[name =
$company]
  return $c//partner }
for $item in document("news.xml")//news_item,
    $c in document("company-data.xml")//company
let $partners := partners($c/name)
where contains(string($item), $c/name)
        and some $p in $partners satisfies
        contains(string($item), $p)
        and $item/news_agent != $c/name
return $item
```

Kuva 1.13. Käyttäjän määrittelemää funktiota käyttävä XQuery-ilmaisu.

Kyselyssä määritellään funktio `partners`, joka tekee toisen XQuery-kyselyn, jonka tuottamien XML-elementtien joukon avulla haetaan ne jutut, joissa esiintyy jokin tuon joukon elementeistä. Tulokseksi saadaan dokumentin `news.xml` ensimmäinen `news_item`-elementti (toinen on yrityksen itsensä julkaisema). Kysely on itseasiassa funktiota käyttävä proseduraalinen ohjelma ja vaatii tekijältään vähintään yhtä paljon aikaa ja vaivaa kuin XSLT-kyselyn tekeminenkin.

### 1.4. XSLT:n ja XQueryn käyttö kyselykielenä

XQueryn ja XSLT:n perustana on XPath-lausekkeet, jotka ovat yksinkertaisimmillaan selkeät, mutta saattavat tulla tietyissä tapauksissa laajoiksi ja vaikeasti tulkittaviksi. XSLT:n elementit ja dokumentin tapaan rakennettava muoto helpottavat hieman jäsentämistä XSLT:n kanssa työskennessä. XQuery käyttää FLWR-ohjausrakennetta kontrolloimaan XPath-lausekkeiden tuottamia tuloksia, mutta kielen määrittelyn lähtökohta on liian toteutusorientoitunut näkökulma, mikä johtaa siihen, että käyttäjän on tiedettävä tarkasti ja etukäteen hakemansa tiedon luonne ja rakenne. Tämä on myös XSLT:n heikkous kyselykielenä. Ero on vain siinä, että kun XSLT ei ole alunperin tarkoitettu tiedonhakuun, kun taas XQuery on. Siksi sen lähtökohtana olisi pitänyt olla nykyistä deklaratiivisempi rakenne. XQueryn tämänhetkinen versio perustuu prosessointiin, jossa kielen syntaksin pyrkimyksenä on ohjelmoijien työn

helpottaminen. Tämä ei ole hyvä lähtökohta, vaan tarvitaan erilainen, deklaratiivisempi kieli.

# 2. Rakenteisuus XML-dokumentissa

Ennen varsinaisen kyselykielen esittelyä tarkastellaan XML-pohjaisen informaation yleistä rakennetta ja kyselykielen yhteydessä esiintyvän muuttuja-käsitteen käyttöä rakenteiden yhteydessä.

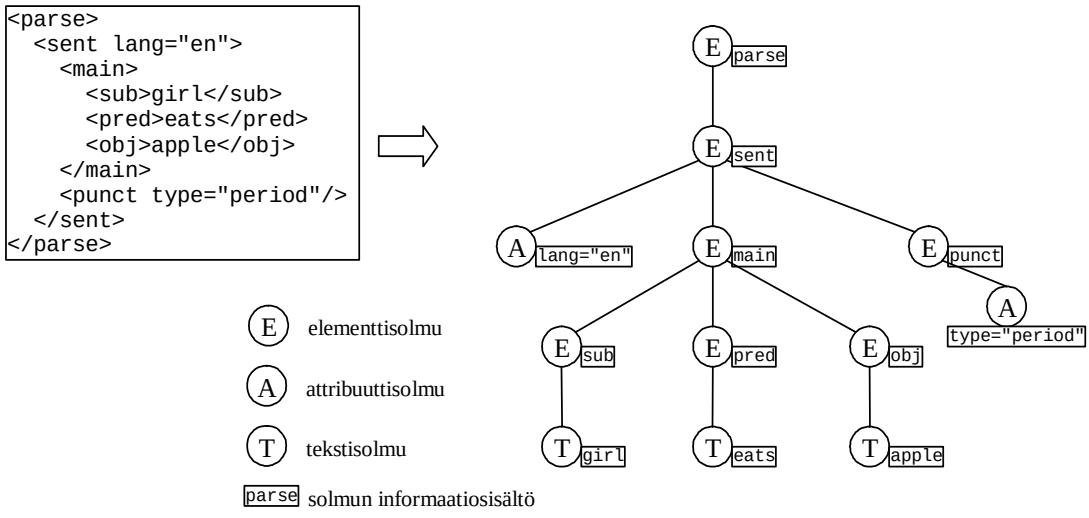
## 2.1. XML-dokumentin rakenteet ja rakenteelliset piirteet

Kuten aiemmin on jo mainittu, XML-dokumentti koostuu johdanto-osasta ja yhdestä juurielementistä. Juurielementti sisältää dokumentin tarjoaman koko informaatiosisällön. Se on rakenteeltaan XML-kielen mukainen elementti, mikä on olennaisin keskeisin looginen rakenneyksikkö XML-informaatiossa. Seuraavissa kohdissa käsitellään tarkemmin XML-elementtiä ja sen rakenteita. Ensin tutkitaan elementin sisäisiä osia ja niiden luonnetta ja sitten elementtien rakennetta suhteessa niiden sisältöön, ja lopuksi elementtien rakenteen vaikutusta koko dokumentin luonteeseen.

### 2.1.1. Elementin loogiset rakenteet

Elementin loogisilla rakenteilla tarkoitetaan tässä niitä osia, jotka ovat merkityksellisiä elementin tarjoaman informaatiosisällön kannalta. Elementti koostuu kolmenlaisesta informaatiosta: nimestä, attribuuteista ja sisällöstä. Sisältö puolestaan koostuu tekstimuotoisesta tiedosta, mahdollisista alielementeistä tai molemmista.

Jos dokumentti ja sen elementit esitetään puumaisena tietorakenteena, puussa kolmentyyppisiä solmuja, eli elementti-, attribuutti- ja tekstisolmuja. Elementtisolmulla voi olla jälkeläissolmuja, kun taas kaksi jälkimmäistä tyyppiä ovat lehtisolmuja. Solmutyyppien erot voidaan esittää myös siten, että elementtisolmun sisäinen informaatio koostuu elementin nimestä. Tekstisolmulla ei ole nimeä, mutta se sisältää tekstiä, attribuutilla taas on sekä nimi- että teksti-informaatiota. [Buneman et al, 2002] Kuvassa 2.1 on esitetty erään XML-dokumentin puurakenne.




Kuva 2.1: XML-dokumentin esittäminen puurakenteena.

Yksinkertaisimmillaan elementti sisältää ainoastaan oman nimensä. Tällainen elementti on tyhjä eikä sillä ole attribuutteja. Nimen tuoman informaation lisäksi voidaan ajatella sen merkityksen olevan sen sijainnissa eli suhteessa muihin dokumentin rakenteisiin. Tämä johtuu siitä, että elementin sisältö on järjestetty joukko elementtejä ja/tai rakenteetonta tekstiä. Elementin sisältämien attribuuttien joukko taas ei välttämättä vaadi järjestyksen säilyttämistä, vaan se on tavallaan valmiiksi sijoitettu tiettyyn paikkaan dokumentissa, isäntäelementtinsä yhteyteen.

Elementti koostuu kolmesta luonteeltaan toisistaan poikkeavasta tiedosta. Nimi on luonteeltaan atominen eli sitä ei voi jakaa osiin[2]. Attribuutit ovat rakenteeltaan avain-arvopareja, joilla voidaan ilmaista esimerkiksi elementtiin liittyvää lisäinformaatiota, jota ei ole katsottu aiheelliseksi sisällyttää itse sisältöosaan. Elementin kolmas tieto on sisältöosa eli alirakenne, joka voi koostua pelkästään tekstielementistä (eli rakenteettomasta tekstistä) tai alielementeistä tai molemmista. Seuraavaksi esitellään tapa luokitella elementtejä alirakenteen perusteella.

### 2.1.2. Rakenteelliset elementtityypit

Dokumentin rakenteen suunnitteluvaiheessa Nykänen jakaa kirjassaan [2001] elementit kolmeen tyyppiin, jotka ovat sisältö, rakenne ja esitystapa. Jälkimmäisin tyyppi voidaan unohtaa, sillä sen esittämä informaatio tulisi kyetä oikeastaan johtamaan kahden ensimmäisen tyyppiluokan muodostamien

[2] Tässä tarkoitetaan nimellä ns. lokaaliosaa [esim. Nykänen, 1999]: XML:ssä on mahdollista käyttää eksplisiittisen nimiavaruuden nimeä, joka merkitään kaksoispisteellä erotettuna lokaaliosan eteen.

elementtien pohjalta. Kahdesta jäljelle jäävästä tyypistä rakennetyyppisillä elementeillä ei yleensä ole tekstielementtejä. Ne ovat mahdollisesti toistuvia ja monia alielementtejä sisältäviä. Näiden tarkoitus on tiedon rakenteen esittäminen, ja niiden varsinainen tietosisältö on alielementeissä.

Sisältöelementtejä käytetään kuvaamaan jotakin tiettyä asiaa, kuten nimeä, hintaa tai muuta sellaista tietoa, jota ei voida tai ei ole haluttu jakaa pienempiin osiin. Tämä tarkoittaa siis dokumenttipuun elementtisolmua, jossa lapsisolmuina ei ole elementtisolmuja. Kuvan 2.2 XML-dokumentin osassa elementit `book` ja `author` ovat siis rakennetyyppisiä, ja elementit `title`, `first`, `last`, `publisher` ja `price` taas sisältötyyppisiä. Tyhjistä elementeistä voitaneen ajatella, että ne ovat sisältötyyppisiä, koska niillä ei ole alielementtejä. Niiden tarjoama informaatio on elementin nimi.

```
...
<book year="2000">
  <title>Data on the Web</title>

<author><last>Abiteboul</last><first>Serge</first></author>
  <author><last>Buneman</last><first>Peter</first></author>
  <author><last>Suciu</last><first>Dan</first></author>
  <publisher>Morgan Kaufmann Publishers</publisher>
  <price>39.95</price>
</book>
```

Kuva 2.2: Rakenne- ja sisältötyyppisiä XML-elementejä.

### 2.1.3. Dokumentti- ja tietokeskeinen rakenne

XML-kielen sovelluksia tai niiden kielioppeja voidaan sanoa luotavan karkeasti ottaen kahdella eri tavalla käyttötarkoituksesta riippuen. XML-kielioppi voi olla luonteeltaan joko dokumentti- tai tietokeskeinen [Fuhr&Grossjohann, 2001]. Dokumenttikeskeisessä lähestymistavassa keskitytään loogisen rakenteen esittämiseen elementtien avulla, jolloin varsinaisen tiedon esitys jää sisältöosan merkitsemättömän tekstin varaan. Dokumentin sisältötyyppiset elementit saattavat olla suhteellisen suurikokoisia, vaikka seassa tietenkin saattaa esiintyä myös pienempikokoisia sisältöelementtejä, joiden luonne on usein suurempien kokonaisuuksien tukeminen. Tämä sopii parhaiten tallennettaessa kirjallisuutta XML-muotoisena, jolloin luvut, kappaleet ja muut sellaiset rakenteelliset osaset kääntyvät pääosin rakenteellisiksi XML-elementeiksi, joiden avulla luodaan dokumentille järjestetty rakenne. Kuvassa 2.3 hahmotellaan tämänkaltaista dokumenttia (kolmea pistettä käytetään jälleen merkitsemään muuta sisältöä):

```
<book>
    <author>Sam Smallwood</author>
    <title>Sam and Fast Money</title>
    <intro>
        <para>Tired to work all day? I have a solution! Follow
the
        instructions of this book<footref id="1"/> and...</para>
        ...
        <footnote id="1">Available as XML file.</footnote>
        ...
    </intro>
    <chapter>
        <title>How to start?</title>
            <section>
                <para>The very first thing to do is...</para>
                ...
            </section>
            ...
    </chapter>
```

Kuva 2.3: Dokumenttikeskeinen XML-dokumentti.

Kuvan esimerkin `para`-elementtien voidaan olettaa sisältävän suhteellisen pitkiä tekstipätkiä, joiden ohessa saattaa esiintyä lyhyempiä `footref`-elementtejä. (Näiden ajatellaan viittavaan `footnote`-elementteihin.)

Tietokeskeisessä lähestymistavassa elementtejä käytetään esittämään rakenteisessa muodossa esiintyviä tietokokoelmia, jossa elementit muodostavat yksittäisiä tietoalkioita ja -rakenteita. Tällä tavalla voidaan koota esimerkiksi lomaketietoa tai luetteloita tai käyttää XML-dokumenttia tai dokumentteja perinteisen relaatiotietokannan tapaan. Tietokeskeisen dokumentin sisältötyyppiset elementit ovat dokumenttikeskeisen dokumentin elementtejä lyhyempiä ja niiden sisältö saattaa olla tyypitettävissä. Sisällöllä ilmaistaan usein lukuarvo, päivämäärä tai joku muu säännöllisen rakenteen omaava arvo tai tieto. Kuvan 2.4 dokumenttihahmotelma esittää tietokeskeistä dokumenttia kuvitteellisen pankin tilisiirtoihin liittyen.

```
<log bank_id="70568">
    ...
    <event>
        <time zone="GTM+02:00">20:56:02</time>
        <from>123456-1234</from>
        <to>987654-9871</to>
        <sum currency="euro">32.23</sum>
        <msg>Phone bill</msg>
        <ref>00112345612340</ref>
     </event>
     <event>
        <time zone="GTM+00:00">18:59:51</time>
        <from>111111-1111</from>
        <to>987654-9871</to>
        <sum currency="euro">100.00</sum>
        <msg>Some other bill</msg>
        <ref>002111111111110</ref>
     </event>
     ...
</log>
```

Kuva 2. 4: Tietokeskeinen XML-dokumentti.

Kuvan esimerkissä rakennetyyppisen `event`-elementin alielementit ovat sisältötyyppisiä. Lisäksi `event`-elementtejä voitaisiin olettaa toistuvan samanrakenteisina (eli samannimiset alielementit sisältävinä) koko dokumentissa siten, että juurielementin sisältö koostuu `event`-elementtisekvenssistä. Tällä tavalla toteutettu dokumentti voitaisiin periaatteessa muuntaa suoraan perinteisen relaatiotietokannan tauluksi. XML:n avulla voidaan kylläkin ilmaista myös monimutkaisempia tietoalkiomalleja, jolloin relaatiomallin sovittamiseksi jouduttaisiin käyttämään useita relaatioita yhtä dokumenttia kohti. Voitaneen myös todeta, että relaatiotietokantoihin rinnastettaessa tietokeskeisestä XML-dokumentista poimittu elementti saadaan vastaamaan joko relaatiotietokannan taulun yksittäistä riviä (kuten esimerkin `event`-elementti) tai attribuuttiarvoa[3] rivillä (esimerkiksi `from`-elementti), riipuen elementin rakenteellisesta tyypistä ja toistuvuudesta.

Tämän luvun loppupuolella annetaan esimerkkidokumenttitietokanta, joka on pyritty konstruoimaan siten, että dokumenteissa ei toisteta samanrakenteisia osia. Dokumentit ovat käyttötapansa ja esittämänsä tietosisällön perusteella suunniteltu edellä esitettyjen rakenteellisten periaatteiden pohjalta.

## 2.2. Esimerkkidokumenttitietokanta

Tässä kohdassa esitellään tutkielmassa käytettävä dokumenttitietokanta ja annetaan dokumenttityyppien tulkintamallit. Kuten aiemmin on esitetty, tulee

[3] Tässä tarkoitetaan relaatiomallin attribuuttia. [esim. Elmasri&Navathe 2000] Sitä ei tule sekoittaa XML:n attribuuttikäsitteeseen.

dokumenttien esittää sellaisia rakenteita, jotka on muodostettu erilaisilla lähestymistavoilla. Näihin dokumenteihin perustuvien esimerkkien yhteydessä esitellään tutkielmassa kehitetyn kyselykielen olennaiset piirteet. Dokumentit kategorisoidaan edellisen kohdan periaatteiden ja niiden sisällön pohjalta.

Dokumenttien kielioppi annetaan DTD-määrittelyn avulla, joka tällä hetkellä on yleisesti käytössä oleva tapa määritellä XML-dokumentin kielioppi. DTD:n merkitsemistapa ei ole XML-syntaksin mukaista. Seuraavaksi annetaan lyhyt kuvaus rakenteessa tarvittavista olennaisista DTD-merkinnöistä käyttämällä apuna kirjaluetteloa kuvaavan dokumentin "bib.xml" DTD-määrittelyä. Dokumentti on esitetty aiemmin.

DTD-määrittely aloitetaan määrittelemällä juurielementti, joka esimerkin dokumentissa on `bib`. Määrittelyssä ilmaistaan elementin sisältö:

```
<!ELEMENT bib  (book* )> .
```

Kuten nähdään, DTD-määritelmä on itse asiassa SGML-elementti, jossa tunnisteena on huutomerkkiä seuraava sana `ELEMENT`. Sitä seuraa sen XML-elementin nimi, jota määrittely koskee (`bib`). Tätä seuraa sulkujen sisässä ilmaistu kieliopin kuvaus. Siinä sanotaan, että `bib`-elementin sisältö koostuu `book`-elementeistä. Asteriksi (`*`-merkki) alielementin nimen perässä merkitsee, että kyseinen elementti voi toistua nolla tai useampia kertoja. Toisin sanoen `bib`-elementti voi olla myös tyhjä. Muita merkintävaihtoehtoja elementtien toistuvuudelle ovat plusmerkki (+), joka tarkoittaa, että elementti voi toistua kerran tai useammin, kun taas kysymysmerkki (?) tarkoittaa nolla tai yhden kerran esiintymistä. Elementin book määrittelyssä käytetään plusmerkkiä seuraavasti:

```
<!ELEMENT book  (title,  (author+ | editor+ ), publisher, price )> .
```

Merkintä tarkoittaa sitä, että elementti `book` sisältää elementin `title`, jota seuraa `author`- tai `editor`-elementit, joita on yksi tai useampi. Sen jälkeen tulevat `publisher`- ja `price`-elementit. Pystyviiva (`|`) erottaa vaihtoehtoiset elementit toisistaan. On huomattava, että merkintä `(author+|editor+)` tarkoittaa sitä, että elementtien on oltava koko ajan samaa tyyppiä. Toisin sanoen `author`- ja `editor`-elementtejä ei voi esiintyä sekaisin samassa `book`-elementissä. Mikäli tämä haluttaisiin mahdollistaa, olisi annettava määrittely

```
<!ELEMENT book  (title,  (author | editor )+, publisher, price )> .
```

Siinä sisempien sulkujen ulkopuolinen plusmerkki tarkoittaa sulkujen sisään jäävän osion toistuvuutta. Sulkuja DTD:ssä voidaan käyttää antamaan vaihtoehtoisia määrittelyjä tai ilmaisemaan jonkin elementtisarjan toistuvuus. Oletetaan, että `book`-elementillä on attribuutti `year`. Se annetaan `ATTLIST`-elementin avulla seuraavasti:

```
<!ATTLIST book year CDATA #REQUIRED > .
```

Elementti määrittelee, että book-elementti sisältää attribuutin nimeltä year, jonka arvo on mikä tahansa merkkijono (CDATA) ja että se on pakollinen (#REQUIRED). Muunlaisista vaihtoehdoista mainitaan myöhemmin tapauskohtaisesti.

Tyypillinen sisältöelementti, jolla ei ole alielementtejä, on elementti title. Se sisältää vain rakenteetonta tietoa, mikä määritellään seuraavassa DTD-määritelyssä:

```
<!ELEMENT title  (#PCDATA )>.
```

DTD-määrittelyssä merkintä #PCDATA tarkoittaa, että sisältö on rakenteetonta tekstiä. Näitä määrittelyjä hyödyntäen esimerkkidokumenteille annetaan ensin niiden DTD-määrittely. Sen jälkeen kuvataan sisältöelementtien käyttötarkoitus ja lopuksi annetaan itse dokumentit.

### 2.2.1. Tietokantamaiset dokumentit

Tietokantamaisella dokumentilla tarkoitetaan tässä tietokeskeistä dokumenttia, jonka sisällön rakenne organisoidaan säännölliseksi. Esimerkkidokumentit ovat "library1.xml" ja "library2.xml", jotka kuvaavat kahden eri kirjaston kirjaluetteloita. Kirjavalikoima niissä hieman vaihtelee ja niiden tiedot organisoidaan hieman erilaisina XML-dokumentteina, esimerkiksi ne sisältävät hieman eri tietoja. DTD-määrittely dokumentille "library1.xml" esitetään kuvassa 2.5.

```
<!ELEMENT bib (book*) >
<!ELEMENT book (title,(author+|editor+),publisher,price)>
<!ATTLIST book year (CDATA) #REQUIRED >
<!ELEMENT author  (last, first )>
<!ELEMENT editor  (last, first )>
<!ELEMENT title  (#PCDATA )>
<!ELEMENT last  (#PCDATA )>
<!ELEMENT first  (#PCDATA )>
<!ELEMENT affiliation  (#PCDATA )>
<!ELEMENT publisher  (#PCDATA )>
```

Kuva 2. 5: Esimerkkidokumentin "library1.xml" DTD-määrittely.

Juurielementillä bib on book-elementtejä, jotka sisältävät informaatiota teoksista. Elementin book attribuutti year ilmaisee teoksen julkaisuvuoden. Elementti title ilmaisee kirjan nimekkeen, ja author-elementti kirjoittajan suku- ja etunimen (last, first). Kirjoittajan sijasta voidaan ilmaista toimittaja(t) (editor). Elementti publisher ilmaisee julkaisijan nimen.

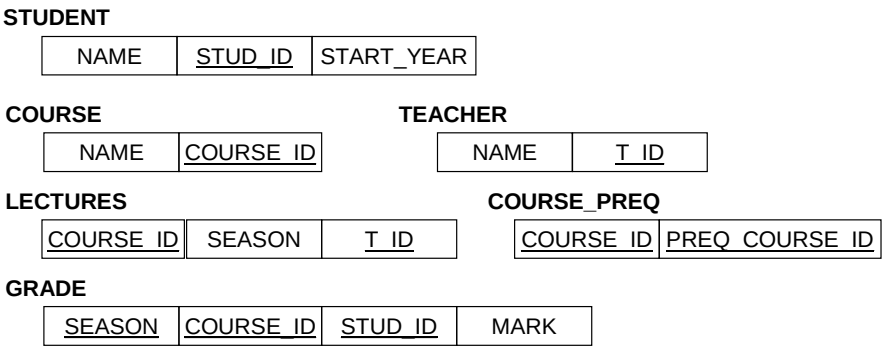
Dokumentin "library2.xml" määrittely esitetään kuvassa 2.6.

```
<!ELEMENT entries (entry*) >
<!ELEMENT entry
(title,pub,class+)>
<!ELEMENT title  (#PCDATA )>
<!ELEMENT pub  (name,year)>
<!ELEMENT name  (#PCDATA )>
<!ELEMENT year  (#PCDATA )>
```

Kuva 2. 6: Esimerkkidokumentin "library2.xml" DTD-määrittely.

Juurielementti `entries` sisältää `entry`-elementtejä, jotka ilmaisevat kirjalle nimekkeet (`title`), julkaisutiedot (`pub`) sekä kirjaston kirjoille antamat luokittelut (`class`), joita voi olla useampia kuin yksi. Luokittelussa käytetään kirjainyhdistelmiä DB ja LP, joista ensimmäinen ilmaisee kirjan sisällön liittyvän tietokantoihin (DataBase) ja jälkimmäinen logiikkaohjelmointiin (Logic Programming). Julkaisutiedot koostuvat julkaisijan nimestä (`name`) ja julkaisuvuodesta (`year`). Täydelliset dokumentit liittyen kuvien 2.5 ja 2.6 DTD-määrittelyihin on annettu kokonaisuudessaan liitteessä 1.

Seuraavat dokumentit liittyvät kuvitteellisen oppilaitoksen arvosanoja sisältävään tietojärjestelmään. Siinä on luotu tietokanta, jossa XML-dokumentit vastaavat ilmeisellä tavalla kuvan 2.7 relaatiotietokannan tauluja.




Kuva 2. 7: Oppilaitoksen relaatiokaavio.

Perusajatuksena on, että relaatiotaulun informaatiosisältöä vastaa XML-dokumentti, jossa toistuvat elementit vastaavat relaatiotaulun rivejä. Tästä poiketen kurssien ennakkovaatimuksia kuvaavan relaation COURSE_PREQ informaatio on sisällytetty relaatiota COURSE vastaavaan XML-dokumenttiin. Samoin arvosanamerkintöjä kuvaava relaatio GRADE on XML-dokumenttiesityksessä sijoitettu relaatiota LECTURE vastaavaan dokumenttiin vaihtelevan kokoiseksi tietokentäksi.

Relaatio STUDENT annetaan dokumenttina "student.xml", jossa elementit organisoidaan kuvan 2.8 DTD-määrityksen mukaisesti.

```
<!ELEMENT students (student*) >
<!ELEMENT student (name,id,start_year)>
<!ELEMENT name  (#PCDATA )>
<!ELEMENT id  (#PCDATA )>
<!ELEMENT start_year  (#PCDATA )>
```

Kuva 2. 8: Dokumentin "student.xml" DTD-määrittely.

Dokumentti kuvaa oppilaita student-elementteinä, jotka kertovat oppilaan nimen (name), tunnuksen (id) ja aloitusvuoden (start_year). Hyvin samankaltainen on myös relaatiota TEACHER kuvaava dokumentti "teacher.xml", jonka DTD-määrittely on kuvassa 2.9.

```
<!ELEMENT teachers (teacher*) >
<!ELEMENT teacher (name,id)>
<!ELEMENT name  (#PCDATA )>
<!ELEMENT id  (#PCDATA )>
```

Kuva 2. 9: Dokumentin "teacher.xml" DTD-määrittely.

Dokumentti sisältää tiedot opettajan (teacher) nimestä (name) ja tunnuksesta (id) (Mahdollisesti dokumentti saattaisi sisältää vielä lisäinformaatiota esim. pätevyyksistä). Relaatio COURSE, joka ilmaisee tarjolla olevat kurssit, on annettu dokumentissa "course.xml". Tähän on lisäksi liitetty ennakkovaatimuksista kertova relaatio COURSE_PREQ, joka löytyy course-elementin preq-alielementistä. Sillä on kuvan 2.10 mukainen DTD-määritys.

```
<!ELEMENT courses (course*) >
<!ELEMENT course (name,preq?,id)>
<!ELEMENT name  (#PCDATA )>
<!ELEMENT preq  (id+)>
<!ELEMENT id  (#PCDATA )>
```

Kuva 2. 10: Dokumentin "course.xml" DTD-määrittely.

Toisin sanoen course-elementti sisältää sisältöelementit name ja id, sekä mahdollisesti preq-elementin. Siinä annetaan yksi tai useampia id-elementtejä, jotka kertovat niiden kurssien tunnukset, jotka ovat edellytyksenä kurssille osallistumiseen. Dokumentissa yhdistetään siis kaksi relaatiota, mikä tapahtuu myös dokumentin "lecture.xml" yhteydessä, johon liittyvä DTD-määrittely annetaan kuvassa 2.11.

```
<!ELEMENT lectures (lecture*) >
<!ELEMENT lecture (course,season,teacher,grades)>
<!ELEMENT course (#PCDATA )>
<!ELEMENT season (year,term)>
<!ELEMENT year (#PCDATA )>
<!ELEMENT term (#PCDATA )>
<!ELEMENT teacher  (#PCDATA )>
<!ELEMENT grades (grade*) >
<!ELEMENT grade (student,mark) >
<!ELEMENT student (#PCDATA) >
<!ELEMENT mark (#PCDATA) >
```

Kuva 2. 11: Dokumentin "lecture.xml" DTD-määrittely.

Dokumentti vastaa relaatiota LECTURES ja GRADES, joissa edellistä kuvaavaan `lecture`-elementtiin sisältyy jälkimmäinen, `grades`-alielementti, johon sisältyvistä `grade`-elementeistä selviää tiettynä lukukautena (`season`) tietystä kurssista oppilaalle annettu arvosana (`mark`). On huomattava, että eri dokumenteissa samannimiset elementit merkitsevät eri asiaa, `course`-, `teacher`- ja `student`-elementit tarkoittavat vastaavien dokumenttien "course.xml", "teacher.xml" ja "student.xml" tarjoamien elementtien `id`-elementtejä. Dokumentit kokonaisuudessaan on esitetty liitteessä 1.

### 2.2.2. Tekstiorientoituneet dokumentit

Tekstiorientoituneet dokumentit ovat pääosin aiemmin mainittua dokumenttikeskeistä tyyppiä, jossa isot sisältöelementit organisoidaan erilaisten rakenneelementtien sisään. Esimerkkeinä käytetään tieteellisten julkaisujen artikkeleita, jotka on organisoitu liitteessä 1 annetun DTD:n mukaisesti. Tarkastellaan rakennetta seuraavaksi.

Artikkelit on merkitty siten, että juurirakenteen `doc` alielementteinä ovat `frontmatter` (tekijä-, julkaisija- yms. tiedot), `docbody` (varsinainen teksti), mahdollisesti `referencelist` (lähdeluettelo) ja mahdollisesti yksi tai useampi `appendix` (artikkelin liite).

Elementillä `frontmatter` on alielementteinä `title` (otsikko), `authorlist` (tekijät), `jname` (mahdollinen lehden nimi, jossa artikkeli on julkaistu), `pyear` (julkaisuvuosi), `vol-issue` (mahdollisen lehden numero), `pages` (mahdollisen lehden sivut), `publisher` (julkaisija), `abstract` (mahdollinen tiivistelmä). Elementti `authorlist` sisältää yhden tai useampia `author`-elementtejä, jotka puolestaan sisältävät etu- (`fn`) ja sukunimen (`sn`) kertovat sisältöelementit sekä henkilöön mahdollisesti liitettävän instituutin (`affiliation`). Elementti publisher koostuu `pname` ja mahdollisesta `paddress`-elementistä, jotka ilmaisevat julkaisijan nimen ja osoitteen.

Dokumentin runko-osa (`docbody`) on jaettu lukuihin (`chapter`). Luvut sisältävät otsikon (`cheading`) sekä joko elementtejä `text` tai `section`, joista

edellinen aloittaa leipätekstin. Jälkimmäinen sisältää (ala) otsikon ja `text`-elementin, joka sisältää joko tekstikappale- (`paragraph`), taulukko- (`table`), tai kuvaelementtejä (`fig`). Tekstinkappaleet voivat sisältää tekstin lisäksi `rid`-elementtejä, jotka ovat viittauksia lähdeluetteloon. Lähdeluettelo (`referencelist`) sisältää viittaustunnisteen (rid), tekijäluettelon (`authorlist`) kuten `frontmatter`-elementtissäkin, lähteen nimen (`rname`) ja julkaisuvuoden (`ryear`) sekä mistä se on peräisin (`rsource`). Dokumentit ja niiden DTD on annettu liitteessä 1.

# 3. Kielen primitiivit

Tässä luvussa esitellään tutkielmassa kehitetyn deklaratiivisen XML-perusteisen kyselykielen tavoitteet ja primitiivit. Siinä tarkastellaan myös kielen rakennetta eli millaisista osista kysely muodostuu.

## 3.1. Muuttujan käyttö XML-pohjaisen tiedon yhteydessä

Kyselykielen lähtökohdaksi on asetettu se, että kyselyiden tulokset esitetään XML-elementteinä, joiden sisältö on peräisin tai johdettavissa joistakin olemassaolevan dokumenttikokoelman elementeistä. Kielen tarjoamien ilmaisujen avulla määritellään oletuksia ja rajoituksia elementin eri ominaisuuksille. Jotta kielen ilmaisusta saataisiin selkeästi yksiselitteinen, on luotava jokin tapa, jolla voidaan viitata tiettyyn tarkasteltavaan elementtiin. Tässä kielessä elementtiin viittaaminen tapahtuu logiikkaohjelmoinnissa käytetyn muuttujakäsitteen avulla. Muuttujalla tarkoitetaan mitä tahansa eksplisiittisesti määrittelemätöntä objektia [esim. Niemi, 2001], joka samaistetaan olemassaolevan dokumenttitietokannan loogisten rakenteiden ilmentymiin perustuen kyselyn ilmaisuihin.

Tutkielman kyselykielessä muuttuja samaistetaan aina johonkin elementtiin. Kuten aiemmin mainittiin, elementti sisältää informaatiota nimestään, attribuuteistaan ja alielementeistään. Tekstityyppinen elementti voidaan laskea omaksi elementikseen, mutta koska elementin attribuutit tai alielementit ovat aina löydettävissä XML-elementin sisällöstä, on luontevaa käyttää XML-elementtiä kuvaavaa muuttujaa. Muuttujaan liitettävän, rajoitetun XPath-ilmaisun avulla voidaan saada kulloisenkin ilmaisun operandiksi tarvittava informaatio. Lisäksi kielellä määritelty kysely saadaan selkeämmäksi, kun tiedetään, että jokainen muuttuja itsessään vastaa XML-elementtiä, eikä esimerkiksi jotakin lukuarvoa tai merkkijonoa. Muuttujan vaikutusalue on koko annettu kyselyilmaus, eli kaikkialla kyselyssä tietty muuttuja viittaa samaan elementtiin.

### 3.2. Kielen tavoitteet

Kieli pyrkii tuottamaan XML-kielen mukaisia elementtejä, joiden sisältämä informaatio on hankittu tunnettun XML-dokumenttikokoelman joukosta. Käyttäjän ei välttämättä tarvitse tuntea dokumenttien rakennetta erityisesti. Elementtejä voidaan hakea niiden muiden ominaisuuksien perusteella, ja ainakin haettavan elementin tai sen sisältävän dokumentin nimen selvittäminen ilman mitään eksplisiittistä informaatiota tulee olla mahdollista. Pyrkimyksenä on tehdä kielestä sellainen, että käyttäjän ei tarvitse osata ohjelmoida tai ymmärtää proseduraalisen ohjelman suoritustapaa. Kielen ilmaisujen on tarkoitus olla deklaratiivisia ja intuitiivisesti tulkittavissa.

Kielessä pyritään tarjoamaan ilmaisut XML-dokumenteissa sekä merkatuille rakenteille että merkkaamattomien tekstiosien käsittelyyn liittyville ilmaisuille. Kielen näkökulmasta tämä tarkoittaa sitä, että siinä tulee olla ilmaisut

1) ulkoisten rakenteellisten ominaisuuksien (esimerkiksi elementin sijainti dokumentissa),
2) sisäisten rakenteellisten ominaisuuksien (esimerkiksi attribuuttien tai alielementtien ominaisuudet) sekä
3) merkkaamattoman tekstisisällön käsittelyyn.

Näistä kaksi ensimmäistä kohtaa on rakenteellisia ilmaisuja ja viimeinen kohta tekstihakuilmaisuja. Rakenteellisten ilmaisujen avulla kyetään esittämään elementin (rakenteellista) suhdetta toisiin elementteihin. Näihin kuuluvat mm. sellaiset ilmaisut, joiden avulla kyetään tarkastelemaan elementtien sisältyvyyttä toisiinsa, sekä niiden järjestystä tai määrää dokumentissa. Esimerkkinä tällaisesta ilmaisusta on "elementin x ensimmäinen `name`-tyyppinen elementti" tai "y on elementin x dokumenttijärjestyksessä[4] toinen välitön alielementti". XPath-kyselykieli on hyvin pitkälle tarkoitettu tämänkaltaisiin kyselyihin ja kielessä onkin käytetty siitä peräisin olevia merkintätapoja ja ilmaisuja. Tekstihakuun tarkoitetut ilmaisut perustuvat merkkijonon tai merkkijonohahmon löytämiseen tekstisisällöstä.

Lisäksi kieli sallii sellaiset ilmaisut, jotka on määritelty tapauskohtaisesti jonkin tietyn sisällön yhteyteen. Esimerkiksi jos dokumenteista haettaisiin oppilaiden suorittamia kursseja (aiemmin esiteltyjen esimerkkidokumenttien mukaisesti) voitaisiin määritellä itse sellainen ilmaisu, joka määrittelee, että "elementti x on arvosana elementille y". Tavallisestihan määriteltäisiin (dokumenttien "student.xml", "course.xml" ja "lecture.xml" tapauksessa) ilmaisu

[4] Dokumenttijärjestys (document order) on se järjestys, jossa elementit ovat XML-dokumentissa.

"elementti x on grade-elementti, joka on lecture-elementissä, jonka course-elementti on sama kuin id-elementti "course.xml"-elementin course-elementissä. Edellä mainitun grade-elementin x student-elementti on id-elementti "student.xml"-dokumentin student-elementissä y". Edut ovat ilmeiset verrattaessa kahta em. ilmaisua keskenään. Tämän lisäksi itsemääritellyt ilmaisut tarjoavat mahdollisuudne transitiivisten ja rekursiivisten ilmaisujen luomiseen siten, että käyttäjältä ei vaadita tämänkaltaisten suhteiden määrittelyä.

Elementtejä käsittelevien ilmaisujen lisäksi kielellä on kyettävä määrittelemään kyselyn tuloksen muoto. Tuloksen määrittelyn on mahdollistettava elementtien nimeäminen uudelleen sekä alkuperäisten rakenteiden uudelleenstrukturointi. Lisäksi tarvitaan tapa, jolla saadaan helposti koottua yhteen eri lähteistä poimittuja elementtejä joiden riippuvuus toisistaan ei ole dokumenttirakenteeseessa vaan kyselyssä ilmaistu. Esimerkiksi oletetaan, että halutaan saada selville dokumentin "library1.xml" kirjojen (book) tekijät (author tai editor) ja heidän kirjoittamiensa kirjojen nimet (title). Dokumentissahan kirjalla on vain yksi nimi, mutta tekijöitä voi olla monta. Tässä yhden suhde moneen on käännettävä päinvastaiseksi. Esimerkiksi XQueryssä tilanne on ratkaistu sallimalla tuloksen yhteydessä määriteltävän sisäkkäisiä kyselyitä [esim. XQuery use, 2002: 1.1.9.4 Q4.]. Tämä on kuitenkin käyttäjän kannalta työlästä ja tarvitaan huomattavasti intuitiivisempi määrittelytapa. Tämä tapa selostetaan myöhemmin kielen syntaksin ja semantiikan esittelyn yhteydessä.

### 3.3. Kielen primitiiveihin liittyvä syntaksi ja semantiikka

Kielen koko syntaksi annetaan liitteessä 2. Sen kaikkein yksityiskohtaisimpia piirteitä ei esitetä tässä. Esimerkiksi, mistä merkeistä aakkosto ja numerot koostuvat. Ennen kielen määrittelyä, selvitetään käytetyn merkintätavan perusperiaatteet.

#### 3.3.1. Käytetystä notaatiosta

Syntaksin ja semantiikan antamisessa on käytetty BNF-notaatiota (Backus-Naur Form), kuten se on määritelty Extended BNF -standardin yhteydessä [ISO/IEC 14977, 1996]. Nonterminaalisymbolin määrittelyssä annetaan ensin symboli, jonka jälkeen seuraa merkki "::=", jonka jälkeen annetaan symbolin vastaava jäsennyssääntö. Kaarisulut ( { ja } ) merkitsevät esityksessä symbolijonoa, joka voi toistua useaan kertaan tai olla esiintymättä lainkaan. Hakasulut ( [ ja ] ) merkitsevät tässä merkkijonoa, joka saattaa esiintyä kerran tai olla esiintymättä lainkaan. Terminaalisymbolit on merkitty lainausmerkkien (") sisään ja ilmaisu päätetään puolipisteeseen (;). Notaatiossa annetaan kahden tai useamman

vaihtoehtoisen merkinnän erottavana merkkinä pystyviiva (|) . Ryhmittelyssä voidaan käyttää sulkeita ( ( ja ) ).

Näkymättömiä merkkejä (rivinvaihto,sarkain,välilyönti) ei ole esityksen selkeyden parantamiseksi merkitty, mutta johtuen toteutustavasta, joka selvitetään luvussa 4, niitä voi esiintyä sulku-, pilkku- ja muita vastaavia välimerkkejä edeltäessä ja seuratessa mielivaltainen määrä tai ei lainkaan. Luonnollisesti kahden erillisen alfanumeerisista merkeistä koostuvan merkkijonon välissä tulee olla yksi tai useampi tyhjä merkki.

### 3.3.2. Kyselyn runko

Kielen perusilmaus on kyselylauseke. Kyselylausekkeen avulla haetaan tunnettujen XML-dokumenttien joukosta kyselyssä määritelty kokoelma XML-elementtejä, joka täyttää kyselyssä määritellyt ehdot. Kyselylausekkeen perusrunko muodostetaan seuraavalla ilmauksella.

query ::=

"get" result_element

"with variables" variables [ "with condition" condition ] "." ;

Kyselyilmaisu alkaa aina varatulla sanalla "get", jota seuraa tulososa eli vastauksen rakenteen määrittely (result_element). Sitä seuraa varatut sanat "with variables", joka aloittaa määrittelyosan. Siinä määritellään kyselyssä käytettävät muuttujat (variables). Ehto-osa (condition) alkaa varatuilla sanoilla "with condition". Sitä ei ole pakko antaa lainkaan, sillä joissain kyselyissä voidaan pelkästään tulos- ja määrittelyosilla tuottaa haluttu tuloselementtikokoelma. Seuraavaksi tarkastellaan ensin tulososaa, sitten määrittelyosaa ja lopuksi ehto-osaa.

### 3.3.3. Tulososa

Tulososa on siis se osa kyselystä, jossa kuvataan kyselyn tuottamat XML-elementit eli kuvataan kyselyn tuloksessa olevat elementit ja niiden keskinäiset suhteet. Kyselyssä määritetään, minkälainen on jokainen tuloksessa oleva elementti. Tulososa määritellään seuraavasti.

result_element ::=

element_name "(" result_token { "," result _token } ")" ;

Määrittely tarkoittaa sitä, että tulokseksi annetaan XML-elementtejä, joiden nimen määrittävä element_name-symboli on tuloselementin nimi[5]. Tuloselementin sisältö ilmaistaan sulkujen sisässä. Siinä erotetaan toisistaan

[5] Kuten aiemmin on mainittu, täydellinen BNF-määrittely on liitteessä 2.

pilkulla elementit, jotka ovat joko muotoilematonta tekstiä tai alielementtien perusteella muodostettuja elementtejä (result_token). Nämä määritellään seuraavasti.

result_token ::= variable_path | text | user_element | group_element;

Vaihtoehtoina ovat muuttujan ja hakupolun perusteella muodostetut elementit (variable_path), merkkaamaton teksti (text), rakenteensa uudelleen ryhmittelevät elementit (group_element) tai elementit, joille käyttäjä määrittelee nimen ja rakenteen kuten tuloselementille (user_element). Näistä kyselyssä tuotettavan informaation kannalta kaikkein olennaisin on muuttujalla viitatut elementit, joiden avulla viitataan dokumenttitietokannassa olevaan informaatioon. Muuttujan ja polun merkintätapa annetaan seuraavalla määrittelyllä.

variable_path ::= variable [ path_step ] ;

Nonterminaalisymboli variable ilmaisee muuttujan, joka on pienellä kirjoitettu käyttäjän valitsema merkkijono (käytännöllisintä on käyttää yhtä merkkiä). Lisäksi muuttujalta vaaditaan, että se on määritelty myöhemmin esiteltävässä määrittelyosassa. Kuten määrittelystä käy ilmi, on elementtipolku (path_step) valinnainen, eli polkua ei tarvitse antaa. Elementtipolulla voidaan kuitenkin valita vain ne osat muuttujan viittaamasta elementistä, jotka kiinnostavat käyttäjää. Polun syntaksi määritellään seuraavassa määrittelyssä.

path_step ::=

path_separator

( path_operator | (element_name [position] | ". ." | "*" | alternate_names)

[ path_step ] ) ;

path_separator ::= "/" | "//" ;

path_operator ::= "#" [ "name" | "doc" | "path"] ;

position ::= "[" ( ("last" ["-" number ] ) | number ) "]";

alternate_names ::= "(" element_name { "|" element_name } ")";

Polku määritetään yksinkertaisimmillaan antamalla polkuerotin (path_separator), joka on joko yksi tai kaksi kauttaviivaa, ja halutun alielementin nimi, tai asteriksi ("*") kuvaamaan minkä tahansa nimistä elementtiä. Elementille voidaan antaa pystyviivalla erotettuja vaihtoehtoisia elementin nimiä, jolloin kyseiseen kohtaan tulosta haetaan vaihtoehtoisia elementtejä. Yhden kauttaviivan merkinnällä haetaan kontekstielementin välittömät alielementit, kahdella välilliset. Kontekstielementillä tarkoitetaan

elementtiä, johon polun kulloinenkin askel liittyy. Esimerkiksi jos ajateltaisiin muuttuja x alustetuksi seuraavaaksi XML-elementiksi

```
<book><title>The Title of The Book</title></book> ,
```

niin kielen mukaisessa alielementtipolkuilmaisussa

```
x/title/#name
```

ensimmäinen polkuaskel on "`/title`", ja tämän polkuaskeleen kontekstielementti on `x` eli esimerkin `book`-elementti. Polkuaskel osoittaa `title`-elementtiin. Seuraava polkuaskel on "`/#name`", jonka kontekstielementti on se elementti, johon polku tätä ennen on edennyt, eli elementti `title`. Merkintä "`#name`" tarkoittaa kontekstielementin nimeä, eli nyt merkkijonoa "title". Kokonaisuudessaan ilmaisu tarkoittaa siis merkkijonoa "title". Ilmaisu "`/#doc`" tarkoittaa sen dokumentin nimeä, jossa kontekstielementti sijaitsee. Ilmaisu "`/#path`" tuottaa juurielementistä lähtien välittömien alielementtien muodostaman kontekstielementin polun.

Jos merkitään pelkkä operaattori "`#`", saadaan kontekstielementin sisältö, eli ilmaisu

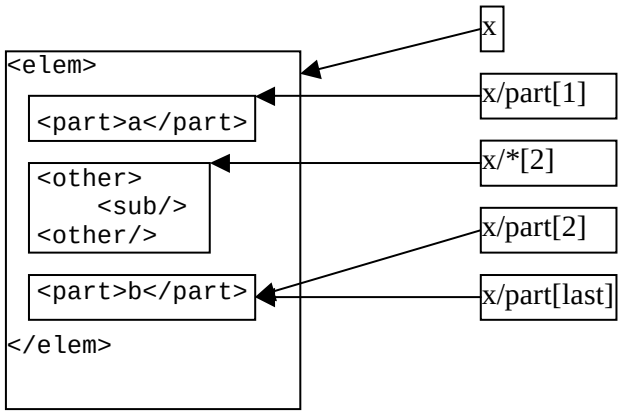
```
x/#
```

tuottaa elementin “`<title>The Title of The Book</title>`”. On huomattava, että mikäli x:n viittaamalla elementillä olisi useampia alielementtejä, viittaisi ilmaisu silloin näiden elementtien joukkoon.

Elementtipolkua voidaan liikkua myös kohti dokumentin juuritasoa käyttämällä elementin nimen sijaan kahden pisteen merkintää. Jos edellämainittu muuttuja `x` viittaisikin esimerkin `book`-elementin sijasta elementtiin `title`, saataisiin book-elementti tulokseen ilmaisulla “`x/..`” .

Tämän lisäksi on mahdollista määritellä positio elementille. Se annetaan elementin nimen[6] jälkeen hakasulkeissa ("[" ja "]"). Positio on joko kokonaisluku yhdestä ylöspäin, jolloin se kuvaa elementin järjestyksellistä sijaintia kontekstielementin vanhemman sisällössä, tai avainsana "last", joka merkitsee viimeistä määritetyn kaltaista elementtiä. Mikäli elementin nimi on määritelty, otetaan järjestyksessä huomioon vain kyseisen nimiset elementit. Viimeisestä edellinen saadaan merkinnällä "last-1" ja niin edelleen. Kuvassa 3.1 esitetään muutama ilmaisu, jotka on yhdistetty nuolella niiden viittamiin elementteihin.

[6] Myös asteriksi on mahdollinen, mutta vanhempi-elementille määrittelyä ei voi antaa, sillä sen positio ei riipu kontekstielementistä.




Kuva 3.1: Esimerkkejä positioilmaisuista.

Tulososan määrittelyssä voidaan haluta käyttää tekstiosia antamaan esimerkiksi jotain kyselyn tekijän esille haluamaa sivuinformaatiota. Tekstielementti (text) annetaan lainausmerkkien (") sisään kirjoitettuna. Tämä muodostaa tuloselementtiin oman tekstielementin. Tätä hyödyllisempi on luultavasti käyttäjän määrittelemä elementti (user_element), joka ilmaistaan seuraavasti:

user_element ::=

element_name "(" result_token { "," result _token } ")" ;

Se on identtinen aiemman result_element -määrittelyn kanssa, eli se muodostaa tuloselementin sisälle käyttäjän määrittelemän alielementin. Esimerkiksi ilmaisu "`user( one("text1"), two("text2") )`" tuottaa tulokseen elementin "`<user><one>text1</one><two>text2</two></user>`".

Ryhmittelevä elementti (group_element) on merkintätavaltaan melko samankaltainen käyttäjän määrittelemän elementin kanssa. Se ryhmittelee sisältämänsä elementit muiden muuten identtisten tuloselementtien kanssa. Ryhmittelevä elementti määritellään ilmaisun

group_element ::=

element_name "{" result_token { "," result _token } "}" ;

avulla. Ainut ero notaatiossa tuloselementtiin ja käyttäjän määrittämään on, että tavallisten sulkumerkkien("(" ja ")") sijasta käytetään aaltosulkuja ("{" ja "}"). Esimerkin avulla pyritään havainnollistamaan näiden merkintöjen eroa. Oletetaan että x viittaa esimerkkidokumenttitietokannan "student.xml"-dokumentin (joihinkin) `student`-elementteihin. Tuloselementtimäärittely

```
result( class( x/name ), x/start_year )
```

tuottaa seuraavan tuloksen.

```
<result>
 <class>
  <name>Joe Jackson</name>
 </class>
 <start_year>2001</start_year>
</result>
<result>
 <class>
  <name>Mark Mills</name>
 </class>
 <start_year>1999</start_year>
</result>
<result>
 <class>
  <name>Peter Gilardi</name>
 </class>
 <start_year>2001</start_year>
</result>
<result>
 <class>
  <name>Robert Poster</name>
 </class>
 <start_year>1999</start_year>
</result>
```

Toisin sanoen tuloksessa on yhteensä neljä `result`-elementtiä, joissa jokaisessa on `class`-elementti ja `start_year`-elementti. Elementti `class` sisältää `name`-elementin, ja `start_year`-elementti on sisältöelementti, jonka sisältö on joko 1999 tai 2001. Käytettäisiin seuraavaa ryhmittelevää kuvausta

```
result( class{ x/name }, x/start_year ).
```

Nyt `class-elementit` ryhmitellään `start_year`-elementin mukaan. Tulos esitettäisiin seuraavasti.

```
<result>
 <class>
  <name>Joe Jackson</name>
  <name>Peter Gilardi</name>
 </class>
 <start_year>2001</start_year>
</result>
<result>
 <class>
  <name>Mark Mills</name>
  <name>Robert Poster</name>
 </class>
 <start_year>1999</start_year>
</result>
```

Toisin sanoen tulos koostuu kahdesta `result`-elementistä, joiden sisältöinä ovat `class`-elementtien sisään ryhmitellyt nimet sekä `start_year`-elementit, joiden perusteella ryhmittely tapahtui. Tällä tavoin voidaan hyvin yksinkertaisella ilmaisulla kääntää myös aiemmin mainittu yhden suhde moneen suhde. Näitä ja muita piirteitä esitellään lisää esimerkkikyselyjen yhteydessä.

### 3.3.4. Määrittelyosa

Määrittelyosalla tarkoitetaan sitä kyselyn osaa, joka nimeää kyselyssä käytetyt muuttujat ja mahdollisesti ilmaisee mihin lähtödokumenttiin ja elementtiin tietty muuttuja viittaa/liittyy. Luonnollisesti muuttujan määrittely koskemaan tiettyä elementtiä vaatii käyttäjältä lähtödokumentin rakenteen tuntemusta, mutta sekä lähtödokumentin että elementin nimen määrittely voidaan sivuuttaa jäljempänä esitettävällä tavalla.

Määrittelyosa on tulososan tapaan pakollinen, sillä kieli vaatii, että muuttujat on erikseen määriteltävä ettei niitä sekoitettaisi ilmaisuiden muihin osiin. Tulos- ja ehto-osassa esiintyvien muuttujien on oltava määrittelyosassa määriteltyjä. Määrittelyosa alkaa varatuilla sanoilla "with variables", jota seuraavan terminaalisymbolin variables syntaksi määritellään seuraavasti.

variables ::= variable_definition { "," variable_definition} ;

Toisin sanoen muuttujamäärittelyilmaisuja (variable_definition) voidaan määrittelyosassa antaa useampi kuin yksi. Muuttujamäärittelyn syntaksi puolestaan on seuraava.

variable_definition ::= variable { "," variable } origin ;

Siinä yksi tai useampi muuttuja voi viitatata samaan kohteeseen (origin), joka puolestaan määritellään

origin ::= "from" "(" document_definition { "," document_definition } ")" ;
document_definition ::= source_doc [ search_path | root_path];
source_doc ::= ("any" | doc_name | "query" "(" query ")" );
search_path ::=
  path_separator search_name { search_path }.
search_name ::= (element_name | "*")[position].
root_path ::= "/".

Toisin sanoen muuttujan hakualue voi olla yksi tai useampi lähtö-dokumentti (source_doc), tai mikä tahansa ("any") saatavilla oleva dokumentti. Valinnaisesti voidaan lähtödokumentin nimen jälkeen antaa hakupolku , jolloin muuttuja viittaa dokumentissa kyseisen polun osoittamiin elementteihin. Hakupolku muodostetaan samalla periaatteella kuin tulososassakin, mutta se lähtee liikkeelle aina dokumentin juurielementistä eikä se salli osoitettavan kuin elementtejä. Pelkän kauttaviivan avulla muuttuja viittaa dokumentin juurielementtiin. Any- ilmauksella ja elementin nimellä voidaan tällä tavalla hakea saatavilla olevien dokumenttien joukosta tietyn nimisiä elementtejä.

Muuttuja voi viitata joko järjestelmän tunnistaman XML-dokumentin tai sisäkkäinen kyselyn tuottaman dokumentin elementteihin. Sisäkkäinen kysely suoritetaan ennen ulompaa kyselyä, ja sen tuottamaa rakenteellista dokumenttia käsitellään kyselyn muissa osissa kuten olemassa olevia XML-dokumentteja. Sen elementteihin viittaavilla muuttujilla ei kuitenkaan ole olemassa polkuoperaattoria (path_operator) "#doc", joka antaa sen dokumentin nimen, josta kyseinen muuttujaelementti on peräisin. On huomattava, että sisäkkäinen kysely on täysin erillinen ulommasta kyselystä ja siinä käytettyjen muuttujien vaikutusalue ei ulotu ulompaan kyselyyn tai päinvastoin.

Pelkästään tulososan ja määrittelyosan avulla voidaan muodostaa kysely. Esimerkkidokumentissa "library1.xml" viimeisenä annetun title-elementin sisältö voitaisiin hakea jommalla kummalla seuraavista kyselyistä.

```
get last_title( x//title[last]/# )
  with variables x from("library1.xml"/ ).

get last_title( x/# )
  with variables x from( "library1.xml"//title[last] ).
```

Ensimmäisessä kyselyssä x alustetaan dokumentin "library1.xml" juurielementiksi bib, josta haetaan kaikkien alielementtitasojen dokumenttijärjestyksessä viimeisintä otsikkoa. Jälkimmäisessä x alustetaan suoraan dokumentin viimeiseksi title-elementiksi. Tulokseksi saadaan seuraava elementti.

```
<last_title>Algorithms + Data Structures = Programs</last_title>
```

Molemmissa kyselyissä otetaan operaattorin "#" avulla title-elementin sisältö ja sijoitetaan se itsemääriteltyyn last_title-elementtiin.

### 3.3.5. Ehto-osa

Ehto-osassa annetaan elementteihin eksplisiittisesti liitettäviä ehtoja, joilla rajoitetaan lisää tulos- ja määrittelyosien sallimaa elementtijoukkoa. Syntaksi ehto-osalle annetaan seuraavaksi.

condition ::= boolean_expression ;

boolean_expression ::= "(" logical_factor { "or" logical_factor } ")" ;

logical_factor ::= ["not"] logical_element { "and" logical_factor};

logical_element ::= primitive | boolean_expression;

primitive ::= primitive1 | primitive2 | primitive3 | primitive4 | primitive5 |
primitive6 | primitive7 | primitive8 | primitive9 | trans_op

Kielessä ehtoilmaisuja yhdistellään käyttämällä Boolen operaatioita and, or ja not. Ilmaisuihin (primitive) liittyy aina totuusarvo. Ilmaisut voivat koskea

elementtien rakenteellisia suhteita, niiden arvoa tai tekstisisältöjä. Lisäksi voidaan määritellä kontekstiin sidottuja ilmaisuita.

**Rakennesuhteiden ilmaisut**

Rakennesuhteiden ilmaisuille on ominaista, että ne ilmaisevat elementtien välisiä tai sisäisiä rakenteellisia suhteita. Kahden elementin välillä suhde voi olla sellainen, että toinen elementtimuuttuja sisältyy toiseen. Tällaiselle suhteelle annetaan seuraava ilmaus.

```
primitive1 ::=
   variable ("has" | "contains") "(" variable_path { "," variable_path} ")" ;
```

Ilmaisussa ensimmäisen mainitun muuttujan viittaama elementti voi sisältää toisien muuttujien ja mahdollisten hakupolkujen viittaamia elementtejä. Sanalla "has" tarkoitetaan välitöntä alielementtiä ja "contains" välillistä. Luonnollisesti jos etsitään välillistä elementtiä, tarkoittaa muuttujan sisältyvyys toiseen sitä, että myös kaikilla sen alielementeillä on sama sisältyvyys isännöivään muuttujaelementtiin.

Toinen olennainen ilmaisu on elementtien järjestys dokumentissa. Ilmaisulla, jossa tutkitaan, seuraako muuttujaelementti toista, on seuraava syntaksi.

```
primitive2 ::=
   variable_path ("precedes" | "before") variable_path ;
```

Tämä suhde on kahden muuttujaelementin tai niiden alielementtien välinen. Ensimmäisen elementin on oltava dokumentissa ennen toista, niin että ne eivät ole sisäkkäisiä (eli ensimmäinen elementti ei sisällä jälkimmäistä). Ilmaisu "precedes" tarkoittaa, että ensimmäistä elementtiä seuraa välittömästi toisena annettu elementti, "before" taas, että välissä voi olla muita elementtejä.

Alielementtien lukumäärää voidaan tarkastella seuraavalla ilmaisulla.

```
primitive3 ::=
  variable_path ("has" | "contains") "elements"
     "(" quantity_token {"," quantity_token}")";
quantity_token ::= element_name ["<" | ">"] number;
```

Ilmaisussa annetaan elementtien nimet ja lukumäärää koskevat ehdot, joiden edellytetään olevan voimassa muuttujan viittaamassa elementissä.

Rakenteellisilla ilmaisuilla voidaan myös testata, onko kyseessä oleva dokumentti mahdollisesti juurielementti tai tyhjä. Nämä voidaan selvittää ilmaisulla.

primitive4 ::=

variable_path "is" ("empty" | "root" | ("type one of" name_list ));

name_list ::= "(" element_name {"," element_name} ")".

Ilmaisussa "is empty" tarkoittaa, että muuttujan (tai sen elementtipolun) viittaama elementti on tyhjä. Varattulla sanalla "is root" ilmaistaan, että elementti on juurielementti. Tässäkin tapauksessa muuttujan elementtipolku on mielekästä säilyttää, voi olla esimerkiksi tarve muodostaa ilmaisu

x/.. is root ,

eli ilmaisu tarkoittaa, että x:n välittömän vanhempielementin on oltava dokumentin juurielementti. (Ilmaisu "x/elem is root" ei sekään automaattisesti ole epätosi, jos jossakin toisessa dokumentissa sattuisi juurielementtinä olemaan täysin vastaava elementti.) Myös ehto-osassa voidaan elementin nimeltä vaatia tiettyjä ehtoja ilmaisulla "is type one of", jota seuraa nimilista. Tällöin muuttujan (tai polun) on viitattava johonkin listassa annetun nimiseen elementtiin.

**Sisällön ominaisuuksien ilmaisut**

Sisällön ominaisuuksilla tarkoitetaan niitä piirteitä, jotka sisältöelementit tarjoavat yhdessä attribuuttien kanssa. Tällaisia ovat yksinkertaisimmillaan ilmaisu, jossa tutkitaan, vastaako elementin sisältö jotain annettua arvoa tai toisen elementin arvoa.

primitive5 ::= variable_path ("=" | "!=") variable_path | value ;

value ::= text | number ;

Ilmaisu tarkoittaa siis, että muuttujan (tai siihen liittyvän hakupolun) viittaaman elementin sisältö on sama ("=") tai ei ole sama ("!=") kuin toisen muuttujan (tai siihen liittyvän polun) viittaaman elementin arvo. Toinen vaihtoehto muuttujalle on eksplisiittinen arvo. Tämä ilmaistaan nonterminaalilla value, joka tarkoittaa joko lainausmerkkien sisään merkittyä merkkijonoa tai numeerista arvoa. Numeerisiin arvoihin liittyen on määritelty lisäksi seuraava ilmaisu.

primitive6 ::= variable_path (">" | "<") variable_path | number ;

Toisin sanoen sillä voidaan vertailla arvoja. Vertailtaessa kahden muuttujan (tai niihin liittyvien hakupolkujen) viittaamia elementtejä on molempien

elementtien sisältöjen oltava numeeriset[7]. Vertailtaessa muunlaisia sisältöjä on ilmaus automaattisesti epätosi.

Transitiivisesti toisiinsa liittyviä elementtejä vertailtaessa käytetään seuraavanlaista ilmaisua.

primitive7 ::= variable_path ">>=" variable_path ;

Tässä ensimmäisen muuttujan viittaaman elementin (tai siihen liittyvän polun) sisällön on oltava välittömästi tai välillisesti sama kuin toisen elementin arvo. Esimerkiksi ilmaisu "x >>= y" tarkoittaa, että on voimassa joko "x = y" tai on olemassa minkä tahansa suuruinen elementtijoukko $\{z_1 , z_2 , ..., z_{n-1} , z_n \}$ siten, että "$x = z_1$" , "$z_1 = z_2$", ... ,"$z_{n-1} = z_n$" ja "$z_n = y$". Myös ilmaisussa annettujen alielementtipolkujen on esiinnyttävä välillisen suhteen muodostavissa elementeissä.

Tekstisisältöä käsittelevät ilmaisut aloitetaan ilmaisulla, joka tutkii, sisältyykö elementin sisältämiin tekstiosiin jonkin toisen elementin tekstiosaa. Tähän kyselykielessä on seuraava ilmaisu.

primitive8 ::=
variable_path ("contains"|"has") "text as"
("in" variable_path) | ("one of" variable_path_list) ;
variable_path_list ::= variable_path {"," variable_path};

Ilmaisussa "has" tarkoittaa välitöntä ja "contains" välillistä suhdetta. Varatulla sanalla "in" tutkitaan yksittäisen elementin sisältöä, kun taas "one of" ilmaisee, että elementin tekstin on sisällettävä (mutta ei välttämättä oltava kokonaan sama kuin) vähintään yhden listassa mainitun elementin sisältämä teksti. Nämä käsittelevät siis ensimmäisen muuttujan viittaaman tekstielementin tekstin osasia. Merkkijonojen on vastattava toisiaan täydellisesti eli myös isojen ja pienten kirjainten erot huomioidaan. Sen sijaan käytettäessä ilmaisua

primitive9 ::= variable_path ( "has" | "contains" ) "text" pattern_list ;

annetaan tekstihahmoihin perustuvia ehtoja välittämättä pienten ja isojen kirjainten eroista. Merkkijonohahmot (pattern_list) määritellään seuraavasti.

pattern_list ::= "(" pattern_token { "or" pattern_token } ")" ;
pattern_token ::= ["not"] pattern_element { "and" pattern_token } ;
pattern_element ::=

[7] Tällainen olisi esimerkiksi elementti <tag>1</tag>. Elementin <tag>1<a/></tag> sisältö taas ei täytä vaatimusta, sillä sisällössä on muita elementtejä.

pattern_text | pattern_list | one_of_textlist | near_list | sim_list ;
pattern_text ::= qt {alphanum | misc | "*" | "?"} qt;

Toisin sanoen tekstihahmot muodostetaan kuten ehtoilmaisuissakin Boolen operaatioilla. Peruselementtinä on merkkijonohahmo (pattern_text), jonka esiintyminen elementissä antaa lausekkeen osalle arvon tosi (ja "not" operaattorin kanssa epätoden). Hahmossa käytetään erikoismerkkeinä asteriksia ja kysymysmerkkiä ("*" ja "?") joista asteriksi merkitsee mitä tahansa minkä tahansa pituista (myös tyhjää) merkkijonoa ja kysymysmerkki yksittäistä mitä tahansa merkkiä. "One of textlist"-ilmaus määritellään seuraavasti.

one_of_textlist ::= "one of" "(" pattern_text {"," pattern_text }")" ;

Ilmaisun mukaan käsiteltävän elementin on toteutettava ainakin yksi listan merkkijonohahmoista. Merkkijonojen välisten sanojen etäisyys toisistaan ilmaistaan seuraavasti.

near_list ::= "near" number "for" "(" pattern_text {"," pattern_text }")" ;

Siinä kaikkien sulkujen välissä esiintyvien hahmojen on löydyttävä numeron osoittaman sanamäärän päässä toisistaan. Sanojen erottimena ovat kaikki alfanumeeristen merkkien ulkopuolelle jäävät merkit. Painotettua tekstihahmonhakua edustaa merkkijonojen samanlaisuutta mittaava ilmaisu

sim_list ::=
[hedge] "sim" "(" pattern_text {"," pattern_text }")" [membership];
hedge ::= "slightly" | "very" | "extremely" | "somewhat" | "indeed" ;
membership ::= number;

Listassa esiintyvien sanojen osumat jaetaan listan pituudella. Tulos, joka on yli 0.5 (50%), kuuluu tulosjoukkoon. Tätä arvoa voidaan säädellä antamalla joukon jäsenyydelle korkeampia tai matalampia arvoja välillä 0-1 tai 0-100 (prosenttia). Lisäksi tulokseen voidaan vaikuttaa "hedge"-operaattorilla, jotka määrittelevät joukkoon kuuluuvuuden tietyn funktion mukaan vaihtelevaksi. Operaatiot pohjautuvat Negnevitskyn kirjan sumeaa logiikkaa käsittelevään lukuun [Negnevitsky, 2001].

**Kontekstisidonnaiset ehdot**

Kontekstisidonnaiset ehdot on tarkoitettu laajentaamaan kyselyilmausten joukkoa siten, että se helpottaisi käyttäjän kyselyn tekemistä jonkin tietyn kontekstin yhteydessä. Nämä kontekstiin sidotut ehdot eivät ole välttämättä loppukäyttäjän itsensä määrittämiä, vaan esimerkiksi järjestelmän ylläpitäjän. Esimerkkinä aiemmin mainittiin määrittely, jossa jokin elementti on jonkin

toisen elementin arvosana. Kontekstisidonnaiset ehdot määritellään omalla määrittelylauseellaan seuraavasti.

rule "==>" notation "." ;

Siinä nonterminaali rule on mikä tahansa käyttäjälle määritelty toiminnallisuus ja notation on se ilmaus, jota kielessä tullaan käyttämään. Notation-osan syntaksi on

notation ::= notation_token {notation_token}
notation_token ::=
( ("elem"|"elemlist"| "list" | "path" )
"(" context_variable ")")|atomic_name.

Kontekstimuuttujien oletetaan vastaavan toiminnallisuussäännön muuttujia. Notaatiossa `elem` merkitsee, että sen tilalle kyselyssä tulee muuttuja, `elemlist` muuttujien pilkulla erotettua listaa, `list` tavallista (esimerkiksi tekstialkioita sisältävää) listaa ja `path` elementtipolkua. Nonterminaali atomic_name on mikä tahansa teksti, jonka avulla ilmaisusta muodostetaan yksilöllinen.

Esimerkkinä annetaan aiemmin mainittu "x on arvosana y:lle" määrittely, olettaen, että toiminnallisuus säännölle is_grade_of(X,Y) on määritelty, seuraavasti:

```
is_grade_of(X,Xp,Y,Yp)==>elem(X)path(Xp) is grade of elem(Y)path(Yp).
```

Tämän jälkeen voidaan kontekstiin sidottua ilmaisua käyttää kuten mitä tahansa ilmaisua. Tätä ilmaisua voitaisiin soveltaa esimerkiksi seuraavasti. (Tässä oletetaan, että tietoa haetaan esimerkkidokumenttitietokannan dokumenteista "student.xml", "course.xml" ja "lecture.xml".)

```
get studies(y/name,grades{c/name,x})
   with variables x from("lecture.xml"/grade),
                  c from("course.xml"/course),
                  y from("student.xml"/student)
   with condition( c/id=x/../../course and x is grade of y).
```

Tulokseksi saadaan kaikkien oppilaiden nimet ja suoritettujen kurssien nimet arvosanoineen.

**Transitiivinen operaattori**

Transitiivisella suhteella tarkoitetaan sellaista suhdetta, jossa objektin suhde toiseen on tosi silloin, kun objektin suhde on tosi mihin tahansa vastaavan suhteen toteuttavaan objektiin, joka välillisesti tai välittömästi on samanlaisessa

suhteessa toiseen objektiin. [Merikoski et al, 1998] Toisin sanoen, suhde x R y on muunnettavissa transitiiviseksi siten, että suhde x $R_{transitive}$ Y on voimassa jos

1) x R y tai

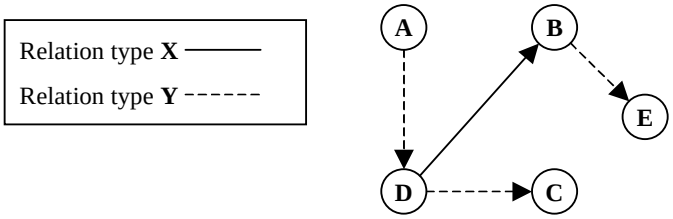
2) on olemassa mielivaltaisen kokoinen joukko $\{z_1, z_2, .. , z_{n-1}, z_n\}$ siten, että on voimassa x R $z_1 \wedge z_1$ R $z_2 \wedge ...$ $z_{n-1}$ R $z_n \wedge z_n$ R y.

Tällaisten suhteiden ilmaisemiseksi annetaan vielä yksi määrittely, joka syntaktisesti kelpaa mille tahansa ehto-osan ilmaisulle, mutta ehdon mielekkyyden kannalta se vaatii, että ehtoilmaisussa esiintyy vähintään kaksi elementtiin viittaavaa muuttujaa. Transitiivisen operaattorin vaikutuksen alainen ilmaisu on tulkittava välilliseksi relaatioksi operaattorin julistaman kahden muuttujan välillä. Näin muodostunut suhde muunnetaan transitiivisella operaattorilla transitiiviseksi seuraavan syntaksin mukaisesti.

trans_op ::= "`transitive`" variable "`to`" variable boolean_expression;

Ilmaisu tarkoittaa, että boolean_expressionin antama kahden muuttujan suhde on transitiivinen. Tosin sanoen esimerkiksi ilmaisut "`transitive x to y (x has y)`" ja "`x contains y`" ovat siis merkitykseltään samat kuten myös "`transitive x to y ( x = y )`" on sama ilmaisun "`x >>= y`" kanssa. Seuraava esimerkki havainnollistaa ilmaisun käyttöä.

Oletetaan lähtödokumentiksi kuvan 3.2 mukainen XML-dokumentti, jolla esitetään saman kuvan suunnattu graafi. Graafissa solmujen väliset yhteydet voivat olla joko tyyppiä X tai Y.




```
<graph>
  <node name="A" rel_y="D"/>
  <node name="B" rel_y="E"/>
  <node name="C"/>
  <node name="D" rel_x="B" rel_y="C"/>
  <node name="E"/>
</graph>
```

Kuva 3.2: Suunnattu graafi ja sitä kuvaava XML-dokumentti.

Kyselyn tulokseksi halutaan ne graafin solmut, joihin on edettävissä solmusta A käyttäen yhteystyyppiä X tai Y. Kyselyilmaisu on esitetty kuvassa 3.3. Siinä määritellään transitiivisen operaattorin avulla, että a:n ja haetun solmun solmun x välinen yhteys on löydettävissä yhteystyyppiä X tai Y käyttäen. Or-operaattorilla määritellään vaihtoehtoiset etenemistavat.

```
get connected_nodes{ x }
   with variables a,x from(any//node)
   with condition(
         a/@name= "A" and
         transitive a to x
               ( a/@rel_y = x/@name or a/@rel_x = x/@name )).
```

Kuva 3.3: Kyselyilmaus kaikkien yhdistettyjen solmujen selvittämiseksi.

Tulokseksi saadaan kuvan 3.4 elementti. Mikäli kyselyssä olisi annettu vain ilmaisu "a/@rel_y = x/@name", tuloksessa olisivat vain solmut C ja D. Transitiivisen operaattorin yhteydessä annettujen muuttujien a ja x välinen suhde määrittyy kaikkien transitiivisuusoperaation sulkujen sisällä olevien lauseiden pohjalta. Tässä esitetyn kaltainen tilanne kyetään näin esittämään ilman rekursiivista ohjelmointia.

```
<connected_nodes>
 <node name="B" rel_y="E"/>
 <node name="C"/>
 <node name="D" rel_x="B" rel_y="C"/>
 <node name="E"/>
</connected_nodes>
```

Kuva 3.4: Transitiivisen kyselyn tulos.

## 4. Kielen toteutus

Tutkielman kyselykielen prototyyppi on toteutettu Prolog-kielellä. Prolog on logiikkaohjelmointikieli, jolla käsitellään objekteja ja niiden välisiä yhteyksiä [Clocksin&Mellish, 1984]. Päinvastoin kuin proseduraaliset ohjelmointikielet, logiikkaohjelmointi on lähestymistavaltaan deklaratiivinen. Toisin sanoen ajatuksena on korvata perinteinen prosessointi ongelman loogisella kuvauksella ja ongelman automaattisella oikeaksi todistamisella. Tämä on erinomainen lähtökohta kyselykielen kehittämiselle.

Lisäksi Prolog tarjoaa erityisesti kielioppien jäsentämiseen ja hahmontunnistukseen tarkoitetun DCG (Definite Clause Grammar)-valmispredikaatin. Tutkielman kielessä on käytetty DCG:tä kielen syntaksin

luomiseen. Seuraavaksi esitellään DCG ja esitetään eräitä sovelluksia ja esimerkkejä DCG:n käytöstä. Lisäksi kerrotaan, miten prototyypin kyselytulkki on rakennettu DCG-kieliopin ympärille.

### 4.1. DCG

Useimmat Prolog-toteutukset tarjoavat kielioppien määrittelyä varten sisäänrakennetun DCG ( Definite Clause Grammar ) -notaation, jolla voidaan käsitellä erilaisten kielten syntakseja. Voidaan ajatella, että DCG tehostaa tavanomaista kieliopin kuvauskieltä, kuten BNF:ää [ISO/IEC 14977, 1996], lisäämällä sille Prologin ilmaisuvoiman.

DCG -toteutukset perustuvat erotuslistojen käyttöön. Prologin sisäinen esitystapa kieliopille a*b*c* (merkkijono, jossa on mielivaltainen määrä merkkejä a, b ja c, tässä järjestyksessä) on seuraavaa Prolog-ohjelmaa vastaava.

```
s(As, Xs ) :-
a(As, Bs ), b(Bs, Cs ), c(Cs , Xs  ).
a( A , A ).
a( A, C) :- connect( A, [a], B),a(B,C).
b( A , A ).
b( A, C) :- connect( A, [b], B),b(B,C).
c( A , A ).
c( A, C) :- connect( A, [c], B),c(B,C).
connect( A, [] , B ).
connect( [X|A], [X|Xs] , B ):-connect(A, Xs, B).
```

Predikaatti `connect/3` voi olla erinäköinen (se ei eksplisiittisesti esitä erotuslistaa, koska sille ei ole suoranaista tarvetta), mutta toimintaperiaate on aina sama. Jokainen nonterminaali tunnistaa oman osansa koko hahmosta, ja antaa lopun, tunnistamattoman osan eteenpäin. Listamuuttujien kuljettaminen koodissa on kuitenkin luettavuuden kannalta hankalaa, joten tämän vuoksi useimmat Prolog-toteutukset tarjoavat DCG -merkintätavan helpottamaan yllämainitun kaltaisten kielioppien muodostamista.

DCG -kielioppisäännöt ilmaistaan Prologissa operaattorilla `-->`. DCG-sääntö "X-->Y" merkitsee "X muodostuu Y:stä" ja "X-->Y, Z" merkitsee "X muodostuu Y:stä, jota seuraa Z" [Clocksin&Mellish, 1984]. Havainnollistetaan tätä seuraavan esimerkin avulla.

```
s --> a , b, c.
a--> [].
a--> [a], a.
b--> [].
b--> [b], b.
c--> [].
c--> [c], c.
```

DCG 1.

Ensimmäisen rivin säännöt luetaan "s muodostuu a:sta, jota seuraa b, jota seuraa c". Seuraava DCG-sääntö luetaan "a muodostuu ei mistään", ja sitä

seuraava rivi antaa vaihtoehdon, että "a muodostuu terminaalisymbolista a, jota seuraa nonterminaalisymboli a". Nämä kaksi sääntöä antavat a:lle siis vaihtoehtoisia muodostamismalleja. DCG-säännöt koskien b:tä ja c:tä luetaan kuten a:ta koskevat säännötkin. Terminaalisymbolit on merkitty Prologin listamerkinnällä hakasulkeiden väliin, ja tyhjä lista ( [] ) merkitsee tyhjää terminaalia.

DCG on voimakas työkalu, koska siinä on mahdollisuus käyttää Prologin piirteitä kielioppisääntöjen yhteydessä, kuten seuraavassa Sterlingin ja Shapiron [1994] kirjasta otetussa, kielioppia DCG 1 laajentavassa esimerkissä esitetään:

```
s( N ) --> a( NA ), b( NB ), c( NC ), { N is NA + NB + NC }.
a( N ) --> [a], a( N1 ),{ N is N1 +1 }.
a( 0 ) -->[].
b( N ) --> [b], b( N1 ),{ N is N1 +1 }.
b( 0 ) -->[].
c( N ) --> [c], c( N1 ),{ N is N1 +1 }.
c( 0 ) -->[].
```

DCG 2 .

Kielioppi DCG 2 ilmoittaa symbolin s argumenttina kyseisen ilmaisun merkkien lukumäärän. Argumentteja voi olla useita, ja niiden avulla pelkän syntaksin ilmaisevaan kielioppiin voidaan lisätä mielivaltainen määrä semanttista merkitystä. Semantiikan lisäksi sallitaan "normaalit" Prologin ilmaisut aaltosulkujen välissä ilmaistuna, joiden avulla tässä tapauksessa suoritetaan yhteenlasku.

Seuraavaksi annetaan esimerkkejä DCG -kieliopeista. Ensin tutkitaan esimerkkiä, joka käsittelee ja kääntää luonnollisen kielen lukusanoja, tämän jälkeen tehdään jäsennin rakenteiselle merkkauskielelle. Käyttötavat eroavat hieman toisistaan.

### 4.1.1. Esimerkki lukusanoista - yksinkertainen kääntäjä

Yksi DCG -sovellusalueista on kielen kääntäminen toiseksi. Koska luonnolliset kielet ovat hyvinkin erilaisia ja pelkästään syntaktisella merkityksellä ei saada koko semanttista merkitystä auki. Aina vastaavuutta ei edes ole toisessa kielessä, tai kuten suomen kielessä, sanajärjestyksellä ei ole merkitystä, keskitymme tässä helpoimpaan mahdolliseen vaihtoehtoon, eli lukusanoihin. Niiden semantiikka on vielä suhteellisen selkeä ja eroavaisuuksien määrä on sopiva tässä esitettävän asian kannalta.

Annamme kieliopin liiteen 3 lukusanoja jäsentävässä ohjelmassa, jossa on määritelty lukusanat välille 0-999 englanniksi ja suomeksi. Toteutuksessa on käytetty Prologin tapaa ilmaista merkkijonot ASCII-koodilistana, eli merkintä `"nolla"` (huomaa lainausmerkit) tarkoittaa viisialkioista listaa

[110,111,108,108,97]. Numerot ovat kirjainten ASCII-koodeja. Lisäksi on annettu predikaatti en_fi_number/3, jonka avulla voidaan käännös toteuttaa. DCG-säännöissä kulkee semantiikkaa tuovina piirteinä lukusanan arvo ja kieli, jolla se on ilmaistu. Esimerkki on muokattu Sterlingin ja Shapiron kirjassa annetun ohjelman pohjalta [1994 (Program 19.9)]. Kokonaisuudessaan se on liitteessä (Liite 3: Lukusanoja jäsentävä ohjelma), tässä käsitellään ohjelman mielenkiintoisimpia osia.

Lukusana ilmaistaan number-säännöllä, jonka argumenttina on lukusanan arvo ja kieltä merkitsevä atomi en (englanti) tai fi (suomi).

```
number( 0, en )-->"zero".       number( 0, fi )-->"nolla".
number( N, Lang )-->xxx(N, Lang).
```

Säännöt kertovat, että lukusana on englanniksi zero ja suomeksi nolla, tai sitten se on vastaavankielinen xxx-ilmaisu. Tämä ilmaisu määrää omanlaisensa ilmaisutavan suomelle ja englannille:

```
xxx(N,en)-->
        digit(D,en)," hundred",rest_xxx(N1,en), {N is D*100+N1}.
xxx(N,fi)-->
        digit(D,fi),{D > 1},"sataa",rest_xxx(N1,fi), {N is D*100+N1}.
xxx(N,fi)-->"sata",rest_xxx(N1,fi), {N is 100+N1}.
rest_xxx(0,Lang)-->[].
rest_xxx(N,en)-->" and ",xx(N,en).
rest_xxx(N,fi)-->xx(N,fi).
```

Nonterminaali digit ilmaisee lukusanan välille 1-9. Koska suomen kielessä ei käytetä ilmaisua "yksisataa", on kyseinen vaihtoehto suljettava pois eli digit-symbolin arvon on suomenkielessä oltava suurempi kuin yksi tai puututtava kokonaan. Säännössä rest_xxx annetaan englannin " and " satojen ja kymmenien välissä, kun taas suomessa lukusana jatkuu suoraan ilman välisanoja tai -lyöntejä. Seuraava sääntö otetaan käyttöön, jos muut xxx-jäsennykset eivät onnistu:

```
xxx(N,Lang)-->xx(N,Lang).
```

Sääntö sallii sen, että sadat voivat puuttua kokonaan. Kymmenlukuja koskevat säännöt ovat

```
xx(D, Lang)-->digit(D,Lang).
xx(T, Lang)-->teen(T,Lang).
xx(N, Lang)-->tens(T,Lang), rest_xx(N1,Lang), {N is T + N1}.
rest_xx(0,Lang)-->[].
rest_xx(N,fi)-->digit(N,fi).
rest_xx(N,en)-->" ",digit(N,en).
```

Tässä ainoa ero kielten välillä on se, että englannissa sanat kirjoitetaan erikseen ja suomessa yhteen. Ilmaisussa luvuille väliltä 10-19 ( teen ) taas

käytetään eri kielille erilaista prosessointitapaa. Englannissa annetaan jokaiselle luvulle oma nimeämissääntönsä:

```
teen(10,en)-->"ten".
teen(11,en)-->"eleven".
teen(12,en)-->"twelve".
etc...
```

Koska suomenkielessä lukua 10 lukuunottamatta kyseiset luvut (10-19) muodostavat lukusanalla alkavan ja sanaan "toista" päättyvän ilmaisun, on tässä tapauksessa mielekkäämpää antaa vain kaksi sääntöä:

```
teen(10,fi)-->"kymmenen".
teen(N,fi)-->digit(N1,fi),"toista",{N is 10 + N1}.
```

Kymmenet ilmaisevassa tens-säännössä kielten välinen ero on samankaltainen. Esitetyn kieliopin käsittelyä helpottamaan tehty Prolog-sääntö

```
en_fi_number(Number,Numero,X):-
        number( X, fi, Numero, [] ),
        number( X, en, Number, [] ),
        name(Nimi,Numero),
        name(Name,Number),
        write('EN: '),write(Name),
        write(' - FI: '),write(Nimi),nl.
```

katsoo, että argumentit Number ja Numero ovat lukuarvoltaan X, ja generoi muuttujille arvot, jollei niitä ole annettu. Järjestelmäpredikaattia name käytetään tässä muuntamaan ASCII-koodilista luettavaan muotoon, joka sitten kirjoitetaan näkyviin.

Seuraavassa esimerkkiajossa katsotaan ensin suoraan number-predikaatilla, mitä tarkoittaa ja millä kielellä on muodostettu sana "kuusisataakaksikymmentaseitseman", ja mitä on englanniksi lukusana "kolmesataaviisikymmentäkaksi" ja sen jälkeen generoidaan lukuarvon 111 ilmaisut kummallekin kielelle.

```
| ?- number( X, D , "kuusisataakaksikymmentaseitseman", [] ).
X = 627 ,
D = fi
| ?- en_fi_number(EN,"kolmesataaviisikymmentakaksi",X).
EN: three hundred and fifty two - FI: kolmesataaviisikymmentakaksi
EN =
[116,104,114,101,101,32,104,117,110,100,114,101,100,32,97,110,100,32,
102,105,102,116,121,32,116,119,111] ,
X = 352
| ?- en_fi_number(EN,FI,111).
EN: one hundred and eleven - FI: satayksitoista
EN =
[111,110,101,32,104,117,110,100,114,101,100,32,97,110,100,32,101,108,
101,118,101,110] ,
FI = [115,97,116,97,121,107,115,105,116,111,105,115,116,97] ;
```

Tässä tapauksessa generatiivinenkin piirre on täysin kontrolloitavissa, sillä tämän nimenomaisen kieliopin avulla voidaan tuottaa täsmälleen tuhat erilaista vaihtoehtoa määriteltyä kieltä kohti. Shapiron ja Sterlingin [1994] mukaan DCG:n generatiivinen piirre ei kuitenkaan ole yleisesti käyttökelpoinen, sillä usein kieliopit sisältävät rekursiivisia sääntöjä, jotka sallivat päättymättömän ketjun toistuvaa ilmaisua. Esimerkiksi aiemmin annettu kielioppi DCG 2.1 on tällainen, se generoi ainoastaan jatkuvasti kasvavaa listaa merkeistä c. DCG 2.2-kielioppia ei voida toteutustavasta johtuen generoida lainkaan. ( Vaikkakin tämä voidaan kiertää ns Iterative Deepening Search-menettelyllä, liitteessä 3. Tämä ei kuitenkaan muuta sitä tosiasiaa, että mahdollisten muodostettavien merkkijonojen joukko on ääretön.) Kuitenkin edellä annetun kaltaisten kielioppien avulla kääntäminen on hieman eri asia, koska siinä semantiikka tulee annetuksi argumentissa, jolloin muodostettava ilmaisu on rajallinen. Lisäksi kieliopin noterminaalit päättyvät aina rajalliseen määrään vaihtoehtoja, ts. päättymättömän rekursion mahdollisuutta ei ole, mikä tietenkin tekee mahdollisten vaihtoehtojen joukkosta äärellisen.

### 4.1.2. XML-jäsennin

Seuraavaksi rakennetaan yksinkertainen XML-jäsennin. Tässä esitellään jälleen vain asiaanliittyviä osia jäsentimestä, koko DCG -määrittely on annettu liitteessä 3.

XML-spesifikaatiossa [XML, 2000] annetaan seuraavanlainen BNF-notaatio XML-dokumentille:

```
document    ::=    prolog element Misc*  .
```

Tässä hieman "oikaisemme" ja määritämme `prolog`- ja `Misc`- symbolien olevan ainoastaan tyhjiä merkkejä (joita ne toki voivat olla), joten keskitymme `element` -osan sisältöön. Tälle annetaan seuraava BNF-määrittely.

```
element    ::=    EmptyElemTag   | STag content ETag
EmptyElemTag    ::=    '<' Name (S Attribute)* S? '/>
STag    ::=    '<' Name (S Attribute)* S? '>'
Attribute    ::=    Name Eq AttValue
ETag    ::=    '</' Name S? '>'
```

BNF 3.1.

Määrittelystä muodostetaan seuraavat DCG-säännöt:

```
xmlelement(element(X,As,[]))-->
        "<",name(X),attributes(As),s(*),"/>".
xmlelement(element(X,As,Es))-->
        "<",name(X),attributes(As),s(*),">",
        content(Es),
        "</",name(X),">".
```

DCG 3.1.

DCG-määrittely on lähes suora käännös BNF -notaatiosta, lisänä on kuitenkin argumentit, joiden avulla muodostetaan semanttinen esitys XML-elementistä Prologia varten. DCG-määrittelyssä vaaditaan myös alku- ja lopputagin samannimisyys. Tätä BNF-notaatio ei kykene ilmaisemaan.

XML -elementillä voi olla attribuutteja, jotka muodostavat aina avain-arvo -parin, jossa avainta seuraa yhtäsuuruusmerkki ja lainausmerkkien sisään sijoitettu arvo. Elementillä ei voi olla montaa samannimistä attribuuttia, mikä on jälleen kerran BNF -notaatiolla mahdotonta esittää. DCG-toteutuksessa se ei ole mikään ongelma:

```
attributes([])-->[].
attributes([A|As])-->s(+),attribute(A),attributes(As),
        {not same(A,As)}.
attribute(A)-->
        name(X),s(*),eq,s(*),qt,non_qtchars(S),qt,
        { name(Av,S),A =.. [X,Av] }. .
```

Predikaatin same/2 negaation avulla rajoitetaan attribuuttien nimivaihtoehtoja. Attribuutit kootaan Prologia varten yksialkioisiksi faktoiksi.

Sisältönään XML -elementtillä voi olla toisia elementtejä tai tekstiä. Seuraavan määrittelyn avulla voidaan ilmaista tämä asia.

```
content([])-->s(*).
content([X|Xs])-->s(*),xmlelement(X),content(Xs).
content([X|Xs])-->pcdata_string(X),con_element(Xs).

con_element( [] )-->[].
con_element( [X|Xs] )-->xmlelement(X),content(Xs).
```

Nonterminaali `con_element` on määritelty estämään tekstielementtien `pcdata_string` peräkkäisyys, mikä aiheuttaisi tekstinkappaleen fragmentoitumisen yksittäisiksi merkeiksi. Kuten voidaan huomata, syntyy tässä XML-elementin sisällön suhteen päättymätön rekursiivinen rakenne, johon edellisen kohdan lopussa jo viitattiin.

Loput liitteessä 3 esitetyistä säännöistä ovat enemmän tai vähemmän toteutukseen sidottuja tai sellaisenaan selkeästi ymmärrettävissä. Seuraavassa esimerkkiajossa nähdään, millaisen tietorakenteen jäsennin saa aikaiseksi. Merkinnällä ~" esitetään lainausmerkin, jotta se voidaan esittää Prologissa merkkijonon (eli lainausmerkkien) sisällä:

```
| ?- xml(X,"<book read=~"yes~">
             <head>
               <title>Ubik</title>
               <author>Philip K. Dick</author>
             </head>
             <body/>
```

```
            </book>",[]).
X =
element(book,[ read(yes) ],
        [element(head,[],
         [ element(title,[],[
                pcdata(`Ubik`)]),
           element(author,[],[pcdata(`Philip K. Dick`)])]),
        element(body,[],[])]) .
```

DCG- kielioppi tarjoaa todella tehokkaan ja helppokäyttöisen lähestymistavan jäsennysongelmiin. Joissakin tapauksissa, kuten yllä, formaalisti määritellyn kielen yhteydessä riittää pelkästään aiemmin annetun notaation muuntaminen DCG -muotoon. Jos tähän jäsentävään XML -kielioppiin lisättäisiin vielä dokumentin tyyppimäärittelyn validoiva osuus, oltaisiin lähes pelkästään XML -merkintätavan syntaksia määrittelemällä samanaikaisesti myös tehty sen kieliopin oikeellisuuden tarkastamiseen kykenevä ohjelma. Kuitenkin jo tällaisenaan se toimii XML- jäsentimenä.

Liitteessä 3 on annettu välineitä, joiden avulla voidaan LPA-Prologissa lukea XML -tiedosto levyltä ja jäsentää sitä. Yksinkertaisella esityspredikaatilla voidaan tulostaa jäsennyksen tuloksena saatu XML-dokumentin Prolog-kuvaus puuesityksenä. Oletetaan, että em. esimerkki olisi tallennettu hakemistoon `c:\temp` nimellä `doc.xml`. Liitteen 3 XML-dokumentin käsittelyohjelmilla saadaan hakupuu tulostettua seuraavan kyselyn avulla:

```
read_file(' c:\temp\doc.xml',A ),       xml(X, A ,[]),  pr_tree(X,0).
```

Tämä tuottaa seuraavan tulostuksen (muuttujien arvotukset on jätetty pois).

```
book
@read
  head
    title
      #PCDATA
    author
      #PCDATA
  body .
```

Kuriositeettina Prolog-sääntö `read_file` sopii hyvin ylläesitettyjen kaltaisten, merkkijonolistojen avulla toteutettujen DCG -kielioppien käsittelyyn, sillä se käyttää, kuten DCG:kin, erotuslistatekniikkaa tiedoston lukemiseksi merkkijonoksi. Ylläesitetyn kaltaisessa ajossa merkkijonolistan `A` häntä jää `read_file`-tavoitteen jälkeen alustamatta, mutta seuraava tavoiterelaatio `xml/3` samaistaa sen tyjäksi listaksi. Jättämällä siinä tyhjän listan tilalle muuttuja, voitaisiin ajatella jonkinlaista ketjutusta, jossa jäljelle jäävään osaan luettaisiin jostain toisesta tiedostosta mahdollisesti jollekin toiselle kieliopille uutta materiaalia. Tästä ketjutuksesta saattaisi olla jossakin tapauksissa hyötyä.

### 4.1.3. DCG-toteutuksen nopeuttaminen

Laajamittaiset DCG -menetelmällä toteutetut kieliopit saattavat olla käännösajaltaan hitaita verrattuna joihinkin toisiin tapoihin. Seuraavaksi esitetään eräs tapa, jolla prosessointia voidaan nopeuttaa.

Chang ja Yeung [1999] esittelevät DCG-määrittelyn käyttöä matemaattisten lausekkeiden analyysissä. Tutkimus liittyy käsinkirjoitettujen matemaattisten lausekkeiden käsittelyyn, mutta heidän DCG-määrittelynsä saa yksirivisen merkkijonoesityksen tällaisesta käsinkirjoitetusta lausekkeesta. Kuvassa 1 on esitetty Changin ja Yeungin esimerkki siitä, kuinka tällainen käännös tapahtuu.

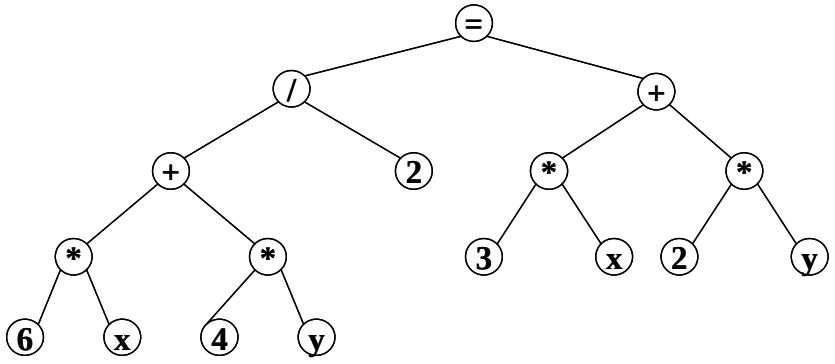
$$\frac{6x-4y}{2} = 3x+2y$$

$$\Downarrow$$

$$(6x+4y)/2 = 3x+2y$$

Kuva 4.1. Kaksiulotteisen yhtälön muuntaminen yksiulotteiseksi.

DCG-sääntöjen avulla muodostetaan yhtälöstä kuvan 2 mukainen puurakenne. Lehtisolmuiksi tulee aina joko luku tai matemaattista muuttujaa kuvaava symboli (x, y jne.). Operaattorisymbolin sisältämällä solmulla on aina alipuut operandeina.




Kuva 4.2. Kuvan 1 yhtälön esitys puurakenteena.

Osaltaan DCG:n tehottomuus verrattuna perinteisiin ohjelmointitapoihin johtuu Prologin toimintamallissa olennaisesta peruutuksesta (backtracking). Annettakoon Chanin ja Yeungin esimerkkiä mukaillen DCG -säännöt

```
expr( [Op,A,B] ) --> term( A ), [ Op ],{ is_valid(Op) }, expr( B ).
expr( A ) --> term( A ).
```

(DCG 4.1),

jotka liittyvät edellä mainittuun matemaattisten lausekkeiden rakenteen analysointiin. Voidaan olettaa, että `term(A)` vaatisi melko paljon prosessointia, sillä se saattaa koostua useista alitermeistä. Ensin on löydettävä kyseinen `term( A )`. Mikäli tämän jälkeen esimerkiksi sääntö `is_valid` epäonnistuu, peruutetaan seuraavaan sääntöön, jossa joudutaan etsimään uudelleen `term(A)`. Chan ja Yeung ratkaisevat ongelman vasemmalle ositetuilla ( left-factoring ) säännöillä. Nyt kielioppi DCG 4.1 esitetäänkin muodossa

```
expr( B ) --> term( A ), more_term( A, B ).
more_term( A, [ Op, A, B ] ) --> [Op], { is_valid(Op) }, expr( B ).
more_term( A, A ) --> [].
```
(DCG 4.2).

Nyt `term( A )` joudutaan prosessoimaan ainoastaan yhden kerran. Kun se on löydetty, käsitellään loppuosa lausekkeesta `more_term( In, Res )` -säännössä, joka antaa lopullisen muodon koko lausekkeen sisällölle. Ideana on lykätä päätöksentekoa niin kauan, että on haarautumiskohdassa ei enää ole löydettävissä samoilla nonterminaaleilla alkavia vaihtoehtoja.

Chan ja Yeung ovat artikkelissaan tehneet kokeita sekä ilman vasemmalle ositusta että sen kanssa. Heidän tuloksissaan on käytetty ajan sijaan mitta-yksikkönä loogisten toimenpiteiden (logical inteferences) määrää, mitä vaaditaan yhtälön jäsentämiseen peruutustekniikan eli DCG:n avulla. Esimerkkinä yksinkertaisemmista lausekkeista heidän tuloksistaan lauseke

$$y = x + \frac{b}{4a}$$

vaatii 89014 toimenpidettä ilman vasemmalle ositusta, sen kanssa toimenpiteitä tarvitaan kertaluokkaa pienempi määrä, 1435 kappaletta. Monimutkaisemmissa lausekkeissa, kuten

$$r = \frac{16ab^2c + 256a^3e - 3b^4 - 64a^2bd}{256a^4}$$

vaaditaan ilman vasemmalle ositusta 19161397 toimenpidettä, mutta sen kanssa vaaditaan ainoastaan 14158 toimenpidettä. Tämä sekä muut esimerkit antavat ymmärtää, että jäsennettävän lausekkeen kompleksisuuden kasvaessa kasvaa suoritusaika eksponentiaalisesti, mutta vasemmalle ositettaessa tämä kyetään minimoimaan tehokkaasti, ja monimutkaisempien lauseiden yhteydessä päästään usemman kertaluokan parannuksiin suoritusajalla mitattaessa.

### 4.1.4. Vasemmalle osituksen soveltamisesta

Esitetyllä tavalla voidaan siis tehostaa DCG-sääntöjen toimintaa jakamalla kielioppi eri tavalla kuin mitä se alunperin on suunniteltu. Toisaalta

esimerkkien DCG 4.1 ja 4.2 välisiä eroavaisuuksia voidaan pitää myös järjestelykysymyksinä. Kuitenkin Changin ja Yeungin esittämien tulosten valossa saavutettu etu on todella merkittävä. Periaatetta voidaan soveltaa yleisesti muissakin yhteyksissä. Esimerkiksi pienellä suunnittelulla edellisen luvun XML-jäsennin oli muutettavissa tehokkaampaan muotoon. Elementin käsittelevä DCG-koodi (DCG 3.1) voidaan osittaa vasemmalle vaikkapa alla olevan esimerkin mukaisesti.

```
xmlelement(element(X,As,Es))-->
        "<", name(X), rest_of_element(X,As,Es).

rest_of_element( X,As,Es )-->
        attributes(As), content_and_rest(X,Es).
%attributes sisältäisi tässä myös viimeisen nonterminaalin s(*)
%käsittelyn

content_and_rest(X,Es)--> ">", content(Es), end_of_element(X).
content_and_rest(X,[])--> "/>".


end_of_element(X) --> "</", name(X), ">".
```

Kun aiempi versio jäsensi ei-tyhjän elementin ensin tyhjän elementin säännöllä, ja terminaalin "/>" sovitus epäonnistui, siirryttiin suorittamaan seuraavaa elementtisääntöä, ja operoitiin uudestaan nimi- ja attribuuttiosa. Vasemmalle ositettaessa nimi- ja attribuuttiosa tunnistetaan kertaalleen, ja nonterminaal `content_and_rest` huolehtii siitä, onko elementti tyhjä vai ei. Jo karkeasti arvioiden voidaan todeta, että vasemmalle ositettuna sääntö on alkuperäistä paljon tehokkaampi. Tosin mitä saavutetaan tehokkuudessa, ehkä menetetään luettavudessa. Suurien tai monimutkaisten käsiteltävien syötteiden yhteydessä tämä on kuitenkin todennäköisesti pieni haitta saavutettuun tehokkuushyötyyn verrattuna.

DCG tarjoaa merkittävän edun perinteisiin ohjelmointimenetelmiin nähden, kun on kysymys hahmontunnistamisesta ja semantiikan analysoinnissa. Tämän lisäksi saadaan generointiominaisuus, joka on hyödyllinen piirre tietyissä sovelluksissa. Seuraavaksi esitellään tutkielman kieltä varten tehdyn kyselytulkin toiminta.

### 4.2. Kyselytulkin toiminta

Kyselykielen toteutuksessa tarvittavat toiminnat jaetaan karkeasti kahteen osaan. Ensimmäisen osan muodostavat toiminnot, joilla muunnetaan XML-dokumentti Prologin ymmärtämään muotoon. Edellä on esitetty tälle DCG-toteutus. Toisen osan muodostaa tulkki aiemmin esitetyille kyselyn ilmaisuille, jotka jäsennetään ja tulkitaan DCG:n avulla Prologin ymmärtämiksi fakta- ja

sääntökokoelmiksi. Ensin kuitenkin esitellään prototyypissä käytetyn XML-dokumentin sisäinen esitystapa.

### 4.2.1. XML:n kuvaaminen Prologin termeiksi

XML-elementin kuvaava termi `element/3` esiintyi jo aiemmassa XML-dokumentin lukemisessa DCG-dokumentiksi. Termin ensimmäinen argumentti on elementin nimi, se on tyypiltään Prologin atomi. Seuraava argumentti on Prologin lista, jonka alkioina ovat elementin attribuutit. Ne on esitetty kaksiarvoisena terminä `(Key,Value)`, jossa `Key` on attribuutin nimi ja `Value` sen arvo. Kolmas argumentti on lista, jonka alkioina on elementin sisältämät alielementit tai tekstiosat. Jälkimmäiset on esitetään termin `pcdata( String )` avulla. Tämän argumentti `String` on LPA-Prolog -kehitysympäristössä [LPA-Prolog] käytettävä merkkijonotietotyyppi. Seuraavan esimerkin avulla havainnollistetaan asiaa.

Oletetaan, että meillä on seuraava XML-elementti.

```
<book org_lang="ru">
  <name>Anna Karenina</name>
  <author>Tolstoi</author>
</book>
```

Tämä kuvataan seuraavaksi Prolog-termiksi (termien funktorit on esitetty paksunnetulla, ja XML-elementistä riippuvat atomit ja merkkijonot on kursivoitu).

```
element(
   book,
   [ (org_lang,ru) ],
   [ element(name,[],[pcdata(`Anna Karenina`)]),
     element(author,[],[pcdata(`Tolstoi`)]) ] ).
```

XML-elementti `book` muuntuu `element`-termiksi, jonka argumentteina ovat elementin nimi (atomi `book`), lista attribuuteista (yksialkioinen lista `[(org_lang,ru)]`), ja lista alielementeistä (kaksialkioinen lista `[element(name,_,_), element(author,_,_)]`[8] ). Alielementti `name` kuvataan element-termiksi, jonka ensimmäinen argumentti on elementin nimi (atomi `name`). Toinen argumentti on tyhjä lista, koska elementillä ei ole attribuutteja. Kolmantena on lista, jonka ainoa alkio on `pcdata`-termi. Tämä kuvaa XML-elementin tekstiosan.

Kokonainen XML-dokumentti kuvataan termillä `xmldoc/3`. Sen ensimmäinen argumentti on järjestelmän tunnistama XML-dokumentin tiedostonimi. Toinen argumentti on lista, jonka alkioina voi olla `xmldecl`, `dtd`, ja `comment`-termejä. Ensimmäinen näistä (`xmldecl`) sisältää argumenttinaan XML-dokumentissa käytetyn version ja merkistön koodin kuvaavan merkkijonon.

[8] Tässä ei luettavuussyistä ole esitetty täydellisiä alielementit kuvaavia termejä.

Toinen (`dtd`) sisältää argumenttinaan DTD-määrittelyn. Termi `comment` on kommenttielementin sisältö (XML-kielessä merkintöjen "`<!--`" ja "`-->`" väliset osat tekstiä ovat kommenteja, joita ei käsitellä rakenteeseen kuuluvina yksikköinä). Näistä mitään ei ole pakko ilmaista, jolloin `xmldoc`-termin toinen argumentti on tyhjä lista. Kolmantena argumenttina on lista, jonka alkioina on dokumentin juurielementin kuvaava `element`-termi sekä mahdollisia `comment`-termejä. Luonnollisesti XML-dokumenttia kuvaava termi `xmldoc/3` siis sisältää kolmannen argumentin listassaan aina vähintäänkin yhden `element`-termin.

### 4.2.2. Kyselykielen ilmaisun käsittely

Tutkielman esittelemän kyselykielen ilmaisut eivät ole Prologin termejä. Ne on tahallisesti muotoiltu sellaisiksi, jotta kieli ei vaatisi käyttäjältä Prologin merkintöjen hallintaa ja että se ei sitoutuisi liiaksi toteutusympäristöönsä.

Toteutettu prototyyppi käyttää kyselyihin DCG-pohjaista tulkkia. Tämä lukee käyttäjän syötteen, tuottaa kyselyn ilmaisujen pohjalta Prologin sääntöjä, yrittää todistaa nämä säännöt ja lopuksi konstruoi ja tulostaa kyselyn tulososan mukaiset XML-elementit. Kyselytulkki ymmärtää kyselylauseiden lisäksi muutaman yksinkertaisen käskyn, kuten "quit", joka lopettaa kyselytulkin käytön.

Tulkki perustuu Prologin `repeat` ja `fail`-predikaattien käyttöön vuorovaikutteisen silmukan aikaansaamiseksi. Minimaalinen ohjelma tällaisen silmukan tuottamiseksi on annettu seuraavassa Prolog-ohjelmassa, joka on mukaelma Sterlingin ja Shapiron kirjan ohjelmasta 12.8 [Sterling&Shapiro, 1994].

```
query:- repeat,read(X),process_it(X),!.
process_it(X):-last_input(X),!.
process_it(X):-write(X),nl,fail.
last_input(quit).
```

Ohjelmassa luetaan predikaatin `read` avulla käyttäjän antama termi `X`. Seuraava tavoite `process_it` epäonnistuu, jos syöte ei ole `last_input`-faktan määrittelemä silmukan keskeyttävä lopetussyöte (eli `quit`). Epäonnistuminen aiheuttaa peruutuksen takaisin tavoitteeseen `repeat`, joka toteutuu ja käyttäjältä kysytään syötettä uudelleen. Tämä toistuu, kunnes käyttäjä antaa termin `quit`. Silloin tavoite `last_input` onnistuu ja sitä seuraava leikkaus (`!`) estää jälkimmäisen säännön testaamisen. Predikaatin `query` määrittelyssä oleva leikkaus estää mahdollisesti jonkin toisessa säännössä tapahtuneen `query`-tavoitteen jälkeisen tavoitteen epäonnistumisen aiheuttaman paluun `repeat`-silmukkaan. Tämä toteutustapa ei ole logiikkaohjelmoinnin kannalta kovin mielekäs, mutta tavanomainen rekursiivinen toteutus aiheuttaisi jossain vaiheessa muistin loppumisen, kun se täyttyisi kyselijäpredikaatin

muistijäljestä. Käytetty tapa pitää Prologin vaatiman muistin määrän pienenä ja on siis lähinnä toteutuslähtöinen.

Minimaalinen vuorovaikutteinen silmukka laajennetaan melko yksinkertaisesti kyselytulkiksi. Seuraava Prolog-ohjelma suorittaa haun pohjautuen käyttäjän syöttämään kyselyyn.

```
query:-repeat,read_user_input(X),process_it(X),!.

process_it([quit]):-!.
process_it(X):-
  phrase( query(q_selection(Selection,Variables,Conditions)), X ),
  findallvars(Variables,Conditions,Allvars),
  findall_tuples(Allvars,Selection,List),
  remove_duplicates(List,PlainRes),
  test_for_reconstruction(PlainRes,Restructured),
  write_result_elements(Restructured,0),fail.
process_it(X):-
  (\+ phrase(query(_),X)),nl,write('Syntax Error.'),nl,fail.
```

Predikaatin query/0 määrittely on itse asiassa lähes identtinen edellisen ohjelman kanssa. Ainut ero on syötteen lukemiseen käytetty predikaatti, tässä ei lueta enää Prolog-termiä, vaan syötetty ilmaisu saadaan termien listana X. Tämä lista jäsennetään phrase/2-valmispredikaatin ja liitteessä 4 annetun DCG-kieliopin avulla. Jos ilmaisu on kielen syntaksin mukainen, tuottaa se termin `query(q_selection(Selection,Variables,Conditions)`. Siinä muuttuja `Selection` on samaistettu tulokseen haluttua elementtiä kuvaavaksi termiksi ja `Variables` on lista, jossa alkioina on kolmiosaisia termejä. Se koostuu kyselyn muuttujista, Prologin elementtiesityksestä sekä lähtödokumentin/dokumentit määrittävästä termistä. Muuttuja `Conditions` on ehto-osan ilmaisujen muodostama tavoite, jonka on toteuduttava `Variables`-muuttujan sisältämien muuttujien viittamien elementtien arvotuksille.

Predikaatin findallvars/3 määrittely on seuraavanlainen.

```
findallvars(Vars,Conds,Allvars):-
   create_find_list(Vars,Vfind),setof(Vars,(Vfind,Conds),Allvars).
```

Siinä create_find_list/2 tuottaa elementtien etsimispredikaateista kootun tavoitteen Vfind, joka yhdessä ehto-osan tavoitteiden (`Conds`) kanssa muodostaa kyselyssä käytettyjen muuttujien arvottamiseksi tarvittavan tavoitteen. Valmispredikaatilla setof/3[9] löydetään kaikki tavoitteen toteuttavat arvotukset muuttujan `Vars` elementeille. Toisin sanoen tavoitteen `findallvars(Vars,Conds,Allvars)` onnistuessa muuttuja `Allvars` on kokoelma XML-elementtejä kuvaavien termien joukkoja (eli lista listoja). Mikäli yhtään tavoitteet toteuttavaa elementtiä ei löydy, on `Allvars` tyhjä lista.

[9] setof/3 käyttäytyy kuten findall/3 -predikaattikin, paitsi että ensiksi mainittu lajittelee kolmannen argumentin tuloslistan aakkosjärjestykseen.

Seuraava predikaatti, findall_tuples/3, konstruoi tuloksen valitsee edellä mainitun listan jokaisesta alkiosta eli elementtikombinaatiosta yhden XML-elementin tulokseen. Tuloselementin sisältö on määritelty tulososan semantiikan toteuttavassa termissä (muuttuja `Selection`), joka on muotoa `selection(Res,Ss)`. Muuttuja Res ilmaisee tuloksen elementin nimen, ja Ss sisällön listaesityksenä. Itse asiassa selection/2 vastaa element/3-termiä, josta on poistettu attribuuttien lista. Predikaatti findall_tuples/3 ja siihen läheisesti kuuluva predikaatti get_selection_element/3 määritellään seuraavasti.

```
findall_tuples(Allvars,selection(Res,Ss),List):-
     setof(X,get_selection_element(selection(Res,Ss),Allvars,X),List).
get_selection_element(selection(Res,Ss),Allvars,element(Res,[],X)):-
     member(OneVars,Allvars),build_simple_tuple(Ss,OneVars,X).
```

Tällä kertaa set_of/3 -predikaattia käytetään tuottamaan lista tuloksena esitettävistä elementeistä. Member/2-predikaatilla valitaan yksi elementtikokoelman listoista. Predikaatin build_simple_tuple/3 avulla valitaan tulokseen halutunlainen elementti. Se määritellään seuraavien sääntöjen avulla (Tulososan muodostama määrittelevä lista ja tuotettava ).

```
build_simple_tuple([],Vars,[]). /*1. sääntö*/
build_simple_tuple([(V,p(Path))|Ss],Vars,[E|Rs]):- /*2. sääntö*/
   member( (V,X,_) , Vars),
   selp_elem(X,Path,E),build_simple_tuple(Ss,Vars,Rs).
build_simple_tuple([pcdata(S)|Ss],Vars,[pcdata(S)|Rs]):-/*3. sääntö*/
   build_simple_tuple(Ss,Vars,Rs).
build_simple_tuple([empty(S)|Ss],Vars,[element(S,[],[])|Rs]):-
   /*4. sääntö*/
   build_simple_tuple(Ss,Vars,Rs).
```

Ensimmäinen argumentti on kyselyn tulos-osan tuottama kuvaus tulos-elementistä. Tuloselementti on valmis, kun rekursion päättävässä ehdossa (1. sääntö) ensimmäisen argumentin listassa ei enää ole jäljellä alkioita. Mikäli tulososassa ilmaistaan muuttuja (2. sääntö) ja mahdollisesti alielementtipolku, kuvautuu se termiksi `(V,p(Path))`, jossa `Path` osoittaa alielementtiin (jos polkua ei ole, `Path` on tyhjä lista). Jos tulososaan on laitettu tekstiä (3. sääntö) tai tyhjä elementti (4. sääntö), muodostetaan niitä vastaavat elementit tuloselementtiin. Monimutkaisemmat tulosrakenteet, eli käyttäjän määrittelemä elementti ja ryhmittelevä elementti, rakentuvat kahden seuraavan säännön avulla.

```
build_simple_tuple(
  [selection(Res,Sel)|Ss],Vars,[element(Res,[],X)|Rs] ):-
     build_simple_tuple(Sel,Vars,X),build_simple_tuple(Ss,Vars,Rs).
build_simple_tuple(
  [m_sel(Res,Sel)|Ss],Vars,[m_element(Res,[],X)|Rs] ):-
     build_simple_tuple(Sel,Vars,X),build_simple_tuple(Ss,Vars,Rs).
```

Ensimmäinen näistä käsittelee käyttäjän määrittelemän elementin ja tuottaa siitä tulokseen elementin. Lisäksi on käsiteltävä kyselyssä annetun elementin sisältö. Toinen sääntö käsittelee ryhmittelevän elementin muuten samalla

tavalla kuin ensimmäinenkin, mutta siinä käytetään elementtiä kuvaavan element/3-termin sijaan m_element/3-termiä, jota tarvitaan myöhemmässä vaiheessa tuloselementtien uudelleenryhmittelyyn.

Jos tuloselementtien joukossa on tulososan tai kyselyn ehtojen määrittelyjen vuoksi syntynyt identtisiä elementtejä, karsitaan ne pois predikaatin remove_duplicates/2 avulla. Näin käsitelty lista muokataan vielä kerran test_for_reconstuction/2-predikaatin avulla. Se ryhmittelee tuloselementtilistan `m_element`-termit uudelleen. Mikäli jotkin elementit ovat `m_element`-termejä lukuunottamatta muuten samanlaiset, sulautetaan ne yhteen ja lisätään niiden `m_element`-termien sisällöt näin syntyneeseen elementtiin. Samalla `m_element` muunnetaan tavalliseksi `element`-termiksi. Seuraavat predikaatit muodostavat uudelleenkonstruoidun tuloksen.

```
test_for_reconstruction(Res,Res):-no_multituples(Res),!.
test_for_reconstruction(Ts,Rts):-
        reconstruct_tuples(Ts,Rets),
        test_for_reconstruction(Rets,Rts).

reconstruct_tuples([],[]).
reconstruct_tuples([Tuple|Ts],[Rtuple|Rts]):-
       reconstruct_tuple(Tuple,Rtuple,Ts,Nts),!,
       reconstruct_tuples(Nts,Rts).

reconstruct_tuple(Tuple,Rtuple,Ts,Nts):-
   find_similar(Tuple,Ts,Sim,Nts),merge_similar(Tuple,Sim,Rtuple).
```

Predikaatin test_for_reconstruction/2 antaa muuttamattoman tuloksen, jos tuloksessa ei ole `m_element`-termejä (no_multituples/1). Muuten etsitään samanlaiset elementit ja sulautetaan ne edellä kuvatulla tavalla yhdeksi elementiksi. Useat sisäkkäiset `m_element`-termit käsitellään yksi hierarkiataso kerrallaan, kunnes no_multituples-predikaatti on tosi.

Lopuksi write_result_elements/2-predikaatti kirjoittaa tuloksen XML-muotoisena. Tämän jälkeen tulee tavoite `fail`, joka aiheuttaa peruutuksen `repeat`-tavoitteeseen, ja seuraava kysely voidaan antaa. Kyselytulkki lopetetaan antamalla syötteeksi kyselyn sijaan sana "quit". Komennolla "do def" päästään antamaan itsemääriteltyjä ilmaisuja. Nämä määrittelyilmaisut käsitellään omalla DCG:llään, jotka tuottavat assert-predikaatilla tallennettavan DCG-määrittelyn.

## 5. Esimerkkikyselyt

Tässä luvussa vertaillaan esimerkkikyselyjen avulla tutkielman kyselykieltä ja XQueryä toisiinsa. Ensin esitellään kuitenkin kyselytyyppejä, joita tarvitaan XML-kielen yhteydessä.

## 5.1. Yleiset kyselykategoriat

Tässä jaetaan kyselyitä muutamiin kategorioihin. Näitä ovat

1) rakenteeseen liittyvät kyselyt,

2) tekstisisältöön liittyvät kyselyt sekä

3) näiden yhdistelmät.

Näillä on muutamia niille tyypillisiä piirteitä, joita selvitetään seuraavissa kohdissa.

### 5.1.1. Rakenteeseen liittyvät kyselyt

Rakenteeseen liittyviä kyselyitä tarvitaan pääasiassa tilanteissa, joissa tunnetaan kyselyn kohteen rakenne, ja tiedetään joillakin rakenteellisilla ominaisuuksilla varustetun elementin sisältävän haluttua informaatiota. Toisaalta muuta rakennetta ei välttämättä tiedetä. Yksinkertaisimmillaan kyselyssä tarkastellaan yksittäistä elementtiä ja sen rakenteellisia ominaisuuksia.

Tarkasteltava rakenne saattaa (rakenteellisesti) liittyä myös johonkin toiseen tarkastelun alaisena olevaan elementtiin. Tällöin kysely on rakenteellisiin tai elementtien välisiin suhteisiin liittyvä. Tähän liittyen rakenteellisen suhteen ei tarvitse välttämättä olla välitön, suhde voi olla myös transitiivinen. Toisin sanoen elementtien välinen suhde ei näy XML-dokumentista suoraan, vaan se on seurausta joidenkin elementtien välillä toistuvasta rakenteellisesta tai sisällöllisestä piirteestä.

### 5.1.2. Sisältöön liittyvät kyselyt

Sisältöön liittyvillä kyselyillä haetaan tietoa, joka saattaa sijaita missä tahansa saatavilla olevien dokumenttien rakenteessa. Käyttäjä ei välttämättä tunne haettua informaatiota sisältävää elementtiä. Yksinkertaisin sisältöön liittyvä kysely on jonkin elementin sisällön kysely. Tätä hieman monimutkaisempia ovat merkkijonohahmojen vertailut, joissa analysoidaan tekstisisällön osia.

### 5.1.3. Yhdistelmäkyselyt

Tämän kategorian kyselyissä halutaan tiettyjä ominaisuuksia sekä sisällöltä että rakenteelta. Tämänkaltaisen kyselyn tekijällä on luultavasti jonkilainen käsitys sekä sisällön luonteesta että elementtien rakenteesta. Yhdistelmäkyselyissä tarkastellaan mm. elementtien sisältöjen ja rakenteiden suhteita sekä ryhmitellään uudelleen tuloksen elementtejä asettamalla tiettyjä oletuksia rakenteiden luonteelle. Tällainen on esimerkiksi jonkin elementin ja sen alirakenteen samankaltainen toistuvuus lähtödokumentissa.

## 5.2. Esimerkkikyselyt

Esimerkeissä esitetään esimerkkikysely sanallisesti, jonka jälkeen annetaan XQueryn ilmaisu ja kehitetyn kyselykielen ilmaisu. Kieliä vertaillaan esimerkkikohtaisesti, ja lopuksi niiden piirteistä esitetään yhteeveto. Esimerkeissä suoritetaan etupäässä rakenteeseen kohdistuvia kyselyjä. Toisin sanoen haetaan tiettyjä elementtejä tietyistä dokumenteista. Tällöin kyselyn tekijän oletetaan tietyssä määrin perehtyneen lähtödokumenttien rakenteisiin. Kyselyt eivät tarjoa kaiken kattavaa valikoimaa XQueryn primitiiveistä ja ne käsittelevät etupäässä niitä piirteitä, jotka kyseisessä kielessä eivät ole tarpeeksi helppokäyttöisiä. Näin ollen tutkielman kyselykieli esiintyy esimerkkien yhteydessä huomattavasti XQueryä edullisemmassa valossa. Kuitenkin tämä on täysin tarkoituksenmukaista, sillä pyrkimyksenä on osoittaa juuri XQueryn heikkouksia. Niiden vuoksi tutkielman kyselykielen kaltaista kieltä tarvitaan tarjoamaan XQuerystä näiltä osin poikkeava kyselykieli.

### 5.2.1. Esimerkkikysely 1

Esimerkki pohjautuu esimerkkidokumenttitietokannan kirjaluetteloita kuvaaviin dokumentteihin "library1.xml" ja "library2.xml". Siinä halutaan etsiä luetteloista löytyvien teosten nimet, toisin sanoen dokumenteista löydettävät `title`-elementit. Vastauksessa löydetyt elementit sisällytetään `result`-elementtiin. Kyselyyn liittyvä XQuery-ilmaisu annetaan kuvassa 5.1.

```
<result>
{
  for $t in
   distinct-values( document("library1.xml")//title or
                    document("library2.xml")//title)
  return
    {$t}
}
</result>
```

Kuva 5. 1. Esimerkkikyselyn XQuery-ilmaisu.

Tulokseen ei haluta samoja `title`-elementtejä useita kertoja. Tämä on estetty XQueryn tähän tarkoitetulla `distinct-values`-funktiolla. Elementin `result` alku- ja lopputunnisteet jäävät tavallaan itse kyselyilmauksen ulkopuolelle. Ne on lisättävä erikseen vastauksena annettavaan XML-dokumenttiin.

XQuery-ilmaisua vastaava tutkielman kyselykielen ilmaus on esitetty kuvassa 5.2.

```
get result{t}
    with variables t from(
       "library1.xml"//title,
       "library2.xml"//title  ).
```

Kuva 5. 2. Esimerkkikysely ilmaistuna tutkielman kielellä.

Tutkielman kielessä oletusarvoisesti tuloksesta poistetaan duplikaatit, jolloin XQuery-ilmaisussa käytettävän kaltaista funktiota ei tarvita. Koska halutaan, että kaikki `title`-elementit ovat yhden `result`-elementin sisällä, käytetään tulososassa kaarisulkeita niputtamaan muuttujan `t` viittaamat `title`-elementit yhteen. Tuloksena saadaan kuvan 5.3 XML-elementti.

```
<result>
 <title>A Guide to the SQL Standard</title>
 <title>Algorithms + Data Structures = Programs</title>
 <title>An Introduction to Database Systems</title>
 <title>Deductive Databases and Logic Programming</title>
 <title>Fundamentals of Database Systems</title>
 <title>Multimedia Database Systems</title>
 <title>Programming in Prolog</title>
 <title>The Art of Prolog</title>
</result>
```

Kuva 5. 3. Esimerkkikyselyn 1 tulos.

Tässä kyselyssä kumpikin ilmaisu on intuitiivisesti melko selkeä. XQueryn ilmaisussa esiintyvä funktio `distinc-values` kuitenkin oireilee viittellisesti XQueryn valtavaa funktiovalikoimaa koskevasta käytettävyysongelmasta. Mikäli käyttäjä haluaa itse määritellä rakenteita vastaukseen, hänen on XQueryssä huolehdittava XML-syntaksin mukaisista alku- ja lopputunnisteiden sijoittelusta. Tutkielman kieli sen sijaan hoitaa tämän automaattisesti tulososan sulkumerkintöjen avulla.

### 5.2.2. Esimerkkikysely 2

Toisessa esimerkissä halutaan tietää dokumentissa "library1.xml" olevat ja vuoden 1990 jälkeen kirjoitettujen teosten nimet (`title`) sekä niiden julkaisijat (`publisher`). Tulokset annetaan `book`-elementteinä `result`-elementin sisässä. XQuery-ilmaisu tälle on annettu kuvassa 5.4.

```
<result>
 {
  for $b in document("library.xml")//book
  where $b/@year > 1990
  return
    <book>
     { $b/title }
     { $b/publisher }
    </book>
 }
</result>
```

Kuva 5. 4. Esimerkkikysely 2 XQuery-ilmaisuna.

XQueryssä käytetään where-osaa ilmaisemaan, että elementin year-attribuutin arvon on oltava suurempi kuin 1990.

Vastaava ilmaus tutkielman kielellä annetaan kuvassa 5.5.

```
get result{ book( b/title, b/publisher ) }
    with variables b from( "library1.xml"//book )
    with condition( b/@year > 1990 ).
```

Kuva 5. 5.Esimerkkikysely 2 tutkielman kielellä.

Kuten edellisessäkin kyselyssä, kaarisulkujen avulla niputetaan tulos yhden result-elementin sisään. Vastaukseksi kysely tuottaa kuvassa 5.6 esitetyn XML-elementin.

```
<result>
 <book>
  <title>A Guide to the SQL Standard</title>
  <publisher>Addison-Wesley</publisher>
 </book>
 <book>
  <title>Deductive Databases and Logic
Programming</title>
  <publisher>Addison-Wesley</publisher>
 </book>
 <book>
  <title>Multimedia Database Systems</title>
  <publisher>Springer-Verlag</publisher>
 </book>
</result>
```

Kuva 5. 6. Esimerkkikyselyn 2 tulos.

Jälleen huomataan, että XQueryssä on kirjoitettava XML-syntaksin mukaiset alku- ja lopputunnisteet. Toisin sanoen käyttäjän on annettava täydellinen tulosdokumentin tunnisteiden merkintä, kun taas tutkielman kielellä annetaan tulosdokumentin rakenne elementtien nimien avulla esitettynä, jolloin tunnisteet muodostetaan automaattisesti.

Tutkielman kielen ilmaisurakenteen eroavaisuus XQueryyn nähden on se, että tutkielman kielessä annetaan tulososa ensimmäisenä, XQueryssä viimeisenä. XQueryssä suoritetaan silmukkaa, jossa `for`- ja `let`-lausekkeilla muuttujiin sijoitetaan arvoja, joista seulotaan `where`-lausekkeilla `return`-osassa käsiteltävät muuttujan arvot. Selkeämmin sanottuna XQuery-ilmaisussa kirjoitetaan sisäkkäisissä silmukoissa esiintyvien muuttujien prosessointiohjeita. Tutkielman kielessä sen sijaan ilmoitetaan haluttu tulos, muuttujien viittauskohteet sekä muuttujilta edellytettävät muut ominaisuudet. Käyttäjän ei tarvitse luoda prosessointia ohjaavia rakenteita tai miettiä muuttujan arvottamisen ja tutkimisen ajallista suoritusjärjestystä.

### 5.2.3. Esimerkkikysely 3

Tässä kyselyssä halutaan selvittää kaikkien dokumenteissa "library1.xml" ja "library2.xml" esiintyvien kustantajien julkaisemat kirjat. Tulokseksi siis halutaan elementti, joka sisältää kustantajan nimen `name`-elementissä sekä kustantajan julkaisemien kirjojen nimet (`title`-elementit) sisältävän elementin `titles`. XQuery-ilmaisu esitetään kuvassa 5.7.

```
for $pub in distinct-values(  document("library1.xml")//publisher or
                              document("library2.xml")//pub/name)
return
    <publisher_data>
        <name>{ $pub/text() }</name>
        <titles>{
            for $b in document("library1.xml")//book or
                      document("library2.xml")//entry
            where
             some $ba in $b/publisher satisfies deep-equal($ba,$pub) or
             some $ba in $b/pub/name satisfies deep-equal($ba,$pub)
            return $b/title
        }
        </titles>
    </publisher_data>
```

Kuva 5. 7. Esimerkki 3 XQueryllä ilmaistuna.

XQueryssä ei ole mitään eksplisiittistä tapaa tuloselementtien ryhmittelyn ilmaisemiseen. Kuvan 5.7 esittämä tapa on ilmeisesti ainut, jolla tämä pystyään tekemään. Kyselyn `return`-osassa on sisäkkäinen kyselyilmaus, jolla kullekin ulomman kyselyn `publisher_data`-elementille haetaan joukko elementtejä sisemmällä kyselyllä. Spesifioinnin kokonaiskuvaa on jo näinkin yksinkertaisessa kyselyssä erittäin hankalaa, ellei mahdotonta hahmottaa. Oman hankaluutensa tuottaa myös dokumenttien tietosisällön sijoittuminen erilaisiin elementteihin. Tämän vuoksi sisemmän kyselyn `where`-osassa on ilmaistava `deep-equal`-ehto kummallekin vaihtoehdolle erikseen.

Tutkielman kielessä tuloselementtien ryhmittely ilmaistaan hyvin yksinkertaisella tavalla, joka on esiintynyt jo aiemmissakin esimerkeissä. XQueryn ilmaisua huomattavasti yksinkertaisempi, mutta saman merkityksen tarjoava kyselykielen ilmaisu esitetään kuvassa 5.8.

```
get data( name( b//(publisher|name)/# ),titles{ b/title } )
    with variables b from( "library1.xml" // book,
                           "library2.xml"//entry ).
```

Kuva 5. 8. Tutkielman kyselykielen ilmaisu esimerkille 3.

Kuvassa 5.8 ryhmittelyn tekniset yksityiskohdat jäävät kyselytulkin tehtäväksi. Käyttäjä vain ilmaisee haluavansa niputtaa yhteen tietyt osat tulokseen tuotettavista elementeistä kaarisulkujen avulla, jolloin ilmaisu ei ole sen vaikeaselkoisempi kuin aiemmatkaan esimerkit. XQueryn sisäkkäisten kyselyiden sekavuuteen verrattuna se tarjoaa huomattavan edun tarjoamalla yksinkertaisen tavan määritellä intuitiivisesti selkeä toimenpide, joka kuitenkin perinteisillä ohjelmointimenetelmillä vaatii (jollaiseksi ilmeisesti XQuerykin on luettava) monivaiheisen prosessoinnin kuvaamisen.

Esimerkkikysely tuottaa vastaukseksi kuvan 5.9 elementit.

```
<publisher_data>
 <name>Addison-Wesley</name>
 <titles>
  <title>A Guide to the SQL Standard</title>
  <title>An Introduction to Database Systems</title>
  <title>Deductive Databases and Logic Programming</title>
  <title>Fundamentals of Database Systems</title>
 </titles>
</publisher_data>
<publisher_data>
 <name>MIT Press</name>
 <titles>
  <title>The Art of Prolog</title>
 </titles>
</publisher_data>
<publisher_data>
 <name>Prentice-Hall</name>
 <titles>
  <title>Algorithms + Data Structures = Programs</title>
 </titles>
</publisher_data>
<publisher_data>
 <name>Springer-Verlag</name>
 <titles>
  <title>Multimedia Database Systems</title>
  <title>Programming in Prolog</title>
 </titles>
</publisher_data>
```

Kuva 5. 9. Esimerkkikyselyn 3 tulos.

### 5.2.4. Esimerkkikysely 4

Seuraavassa kyselyssä etsitään tietoa dokumentista "bicycle.xml". Siinä halutaan saada selville, missä polkupyörämallissa käytetään hiilikuituvannetta (coal fiber rim). Tuloksena halutaan siis `part`-elementin `name`-alielementti, joka alkaa sanalla "`bicycle`". XQueryllä kysely tehdään kuvan 5.10 ilmaisun avulla.

XQueryn ilmaisu perustuu käyttäjän itsensä tekemään rekursiiviseen funktioon `is_part_of`. Ilmaisun tekeminen vaatii jo melko kehittyneiden ohjelmointimenetelmien hallintaa. Käyttäjän on osattava luoda funktio, ja vieläpä rekursiivinen sellainen, mikä ei välttämättä onnistu lainkaan tavallisilta loppukäyttäjiltä. Funktiot eivät itsessään tee kielestä vaikeata, mutta käyttäjän on huolehdittava jälleen hyvin tarkkaan siitä, miten kysely on määriteltävä, vaikka todennäköisesti tärkeämpää olisi antaa määrittely sille, mitä on tehtävä.

```
define function is_part_of($r as element,$p as
element) as boolean
{
   $p/part_id = $r/part_of or
   (some $z in document("bicycle.xml")//part
       satisfies $p/part_id = $z/part_of and
                 is_part_of($r,$z) )
}

<coal_fiber_cycle>
  {
    for $p in document("bicycle.xml")//part
    let $r ::= document("bicycle.xml")//part
    where starts-with($p/name "bicycle") and
          $r/name="coal fiber rim"  and
          ispart_of($r,$p)
    return $p/name
  }
</coal_fiber_cycle>
```

Kuva 5. 10. Esimerkkikysely 4 XQueryllä ilmaistuna.

Kuten aiemmin on esitetty, tutkielman kieli tarjoaa transitiivisten suhteiden käsittelyyn (esimerkin tilanne) omat operaattorinsa, joilla voidaan ilmaista, että muuttujien viittamien elementtien välillä vallitsee transitiivinen suhde. Esimerkin tapauksessa voidaan käyttää joko transitiivista operaattoria (muodossa "`transitive` **`c`** `to` **`r`**`( `**`c`**`/part_id = `**`r`**`/part_of )" ) tai, kuten kuvassa 5.11, transitiivistä yhtäsuuruusmerkkiä ilmaisussa "**`c`**`/part_id >>= `**`r`**`/part_of`".

```
get coal_fiber_cycle( c/name )
   with variables c,r from("bicycle.xml"/part)
   with condition(
        c/name has text( "bicycle*"  ) and
        r/name="coal fiber rim" and c/part_id >>= r/part_of ).
```

Kuva 5. 11. Esimerkkikysely 4 tutkielman kielellä.

XQueryyn verrattuna tutkielman kieli tarjoaa selkeästi deklaratiivisemman tavan ilmaista asia. Käyttäjän ei tarvitse huolehtia muista kuin kyselylle olennaisten muuttujien olemassaolosta. Samaten muuttujien välisten suhteiden ilmaisussa tarvitsee vain ilmaista kahden muuttujan välisen suhteen olevan transitiivinen[10]. Tätä voidaan verrata kuvan 5.10 XQuery-funktioon, jossa joudutaan käyttämään välillä ylimääräistä muuttujaa ilmaisemaan suhteen transitiivisuutta. Kyselyn tuloksena saadaan kuvan 5.12 elementti.

```
<coal_fiber_cycle>
 <name>bicycle mk3</name>
</coal_fiber_cycle>
```

Kuva 5. 12. Esimerkkikyselyn 4 tulos.

### 5.2.5. Esimerkkikysely 5

Tässä kyselyssä haetaan oppilaitoksen tietokantaan liittyvistä dokumenteista opettajat ja ne oppilaat, joille he ovat antaneet arvosanamerkintöjä. Tulokseen halutaan opettajan nimi, opetetut oppilaat ja oppilaan kyseiselle opettajalle suorittamat kurssit. Relaatiotietokantamaisesta rakenteesta johtuen tässä vaaditaan alielementtien vertailuja enemmän kuin aiemmissa kyselyissä. Niillä luodaan liitokset eri elementtien välillä, relaation avainattribuuttien tapaan. XQueryllä kysely ilmaistaan kuvan 5.13 mukaisesti.

[10] Esimerkki tästä on annettu kuvien 3.2 - 3.4 yhteydessä.

```
for $t in document("teacher.xml")//teacher
return
    <result>
        { $t/name }
        <courses>
        {
            for $l in document("lecture.xml")//lecture
            let $c := document("course.xml")//course
            where $c/id = $l/course and
                          $t/id = $l/course
            return
               <course>
                 {$c/name}
                 <students>
                 {
                   for $s in document("student.xml")//student
                   where some $g in $l//grade
                      satisfies $g/student = $s/id
                   return
                      $s/name
                 }
                 </students>
               </course>
        }
        </courses>
    </result>
```

Kuva 5. 13. Esimerkkikysely 5 XQuery-ilmaisuna.

Ilmaisu muodostetaan tässä kolmesta sisäkkäisestä FLWR-lauseesta. Kuten kuvasta näkyy, ilmaisua ei voi pitää intuitiivisesti helposti tulkittavana. Luettavuutta haittaa ennen kaikkea muuttujamäärittelyiden ja -rajoitteiden sekä tuloksen tuottamiseen tarkoitettujen ilmaisuiden hajautuminen ja sekoittuminen eri puolille kokonaisilmaisua. Lisäksi muuttujien eri arvotusten liittyminen toisiinsa on ongelmallista. Tiettyä muuttujan arvotusta kohti voi olla lukuisia muita toisen muuttujan arvotuksia. Esimerkissä tarvittavat elementtien väliset suhteet ovat sellaisenaan monimutkaisia, ja kuvan 5.13 kaltaista ilmaisua suunnitellessa pitää vielä näiden suhteiden lisäksi ottaa huomioon muuttujien oikeanlainen sijoittaminen `for`-,`let`- ja `where`-lauseisiin.

Tutkielman kielessä pidetään tuloksen esittäminen, muuttujien määrittäminen ja elementtien välisten suhteiden kuvaaminen selkeästi erillään toisistaan. Esimerkkikysely on ilmaistu kuvassa 5.14 tutkielmassa kehitetyllä kyselykielellä.

```
get result( t/name, courses{ c/name,studens{ s/name } } )
    with variables t from( "teacher.xml"//teacher ),
                           l from( "lecture.xml"//lecture ),
                           c from( "course.xml"//course ),
                           s from( "student.xml"//student )
    with condition( t/id = l/teacher and
                           c/id = l/course and
                           s/id = l//student ).
```

Kuva 5. 14. Esimerkkikysely 5 tutkielman kielellä ilmaistuna.

Tässä suurin ero XQueryyn on ilmaisun osien selkeä ryhmittely. Tulososa annetaan ensimmäisenä, ja kuten aikaisemmin on todettu, se antaa selkeän mallin tuloselementtien muodolle. Ilmaisussa käytettävien muuttujien viittaamat lähtödokumenttien elementit luetellaan määrittelyosassa, josta tarvittaessa nähdään muuttujien vaikutusalue. XQueryssä muuttujan määrittely voi olla lähes missä tahansa ilmaisun osassa, mikä vaikeuttaa luettavuutta. Tutkielman kielen ehto-osa ilmaisee ne ehdot, jotka vastauksen täytyy toteuttaa. Ehto-osassa annetaan ehtoilmaisut yhdistettyinä Boolen and-operaattoreilla. Tämä SQL-kielen kaltainen määrittelytapa antaa käyttäjälle systemaattisen tavan kyselyjen formulointiin, kun XQueryssa käyttäjä määrittelee vastauksen elementtihierarkioittain kullekin tasolle liittyvine ilmaisuineen. Lisäksi hänen on huolehdittava elementtihierarkioihin liittyvien muuttujien keskinäisestä synkronoinnista. Lienee selvää, että tutkielman kielen tapa on XQuerya huomattavasti käytännöllisempi.

Esimerkin tuottamat tuloselementit on annettu kuvassa 5.15.

```
<result>
 <name>Gibson</name>
 <courses>
  <name>Database Programming</name>
  <students>
   <name>Ephraim Katz</name>
   <name>Paul Jones</name>
   <name>Peter Gilardi</name>
   <name>Robert Poster</name>
  </students>
  <name>Introduction to
Programming</name>
  <students>
   <name>Ephraim Katz</name>
   <name>Mark Mills</name>
   <name>Paul Jones</name>
   <name>Robert Poster</name>
  </students>
 </courses>
</result>
<result>
 <name>Jones</name>
 <courses>
  <name>Data Structures</name>
  <students>
   <name>Ephraim Katz</name>
   <name>Joe Jackson</name>
   <name>Mark Mills</name>
   <name>Paul Jones</name>
   <name>Pavel Stransky</name>
   <name>Peter Gilardi</name>
   <name>Robert Poster</name>
  </students>
 </courses>
</result>
<result>
 <name>Mills</name>
 <courses>
  <name>Logic Programming</name>
  <students>
   <name>Ephraim Katz</name>
   <name>Mark Mills</name>
   <name>Paul Jones</name>
   <name>Pavel Stransky</name>
   <name>Robert Poster</name>
  </students>
 </courses>
</result>
<result>
 <name>Moore</name>
 <courses>
  <name>Database Programming</name>
  <students>
   <name>Mark Mills</name>
  </students>
  <name>Introduction to
Programming</name>
  <students>
   <name>Joe Jackson</name>
   <name>Pavel Stransky</name>
   <name>Peter Gilardi</name>
  </students>
 </courses>
</result>
```

Kuva 5. 15. Esimerkkikyselyn 5 tulos.

### 5.2.6. Esimerkkikysely 6

Seuraavalla esimerkillä haetaan XML-dokumentista "art_001.xml" sellaiset tekstinkappaleet (`text`-elementtejä), joissa on viitattu (`rid`-elementillä)

sellaiseen lähteeseen (ref-elementti), jonka tekijöiden joukossa (authorlist) on Clark.

Kysely suoritetaan XQueryllä kuvan 5.16 ilmaisun perusteella.

```
for $x in document("new/art_001.xml")//text
let $r := document("new/art_001.xml")//ref
where $r/rid = $x/rid and $r/authorlist//* = "Clark"
return
  <clark_refer>
    {$x}
  </clark_refer>
```

Kuva 5. 16. Esimerkkikysely 6 XQueryllä.

Vastaava ilmaisu tutkielman kielellä on annettu kuvassa 5.17.

```
get clark_refer(x)
    with variables x from("art_001.xml"/text),
                   r from("art_001.xml"/ref)
    with condition(  x/rid = r/rid and
                     r/authorlist// * ="Clark").
```

Kuva 5. 17. Esimerkkikysely 6 tutkielman kielellä.

Ilmaisutapojen eroavaisuudet ovat jälleen rakenteessa. Tällä kertaa kuitenkaan erot eivät ole kovin suuria. XQueryn rakenteelle ominainen jäsentymättömyyskin jää tämänkaltaisessa kyselyssä suhteellisen pieneksi. Huomattavaa on, että mikäli tutkielman kielen muuttujat olisi varustettu $-etuliitteellä, voitaisiin ehto-osa siirtää mitään muuttamatta XQueryn where-osaksi. Tämä ei ollut tutkielman kielen suunnitteluvaiheessa tarkoituksellista, vaan perustuu XPath-ilmaisujen kiistattomaan hyödyllisyyteen. Jos suoritettavassa kyselyssä riittää pelkän XPathin ilmaisuvoiman hyödyntäminen, pystytään XQueryn ilmaisustakin saamaan intuitiivisesti tulkittavissa olevaa.

Kysely tuottaa tulokseksi kuvan 5.18 XML-elementin.

```
<clark_refer>
 <text>
  A path expression is an expression involving
node names (tags and attribute names) that
describes
a set of paths in the document tree.
The choice of what language we use to define
path expressions is important to the expressive
power of keys, and there are a number of choices.
In XML-Schema, XPath
  <rid>15</rid>
  expressions...
 </text>
</clark_refer>
```

Kuva 5. 18. Esimerkkikyselyn 6 tulos.

## 6. Yhteenveto ja loppupäätelmät

Tutkielmassa kehitetty kyselykieli esittää vaihtoehdon XML-dokumenttien kyselykieleksi. Siinä on pyritty poistamaan XQueryn kaltaisen kielen hankalia piirteitä. Näitä ovat esimerkiksi ilmaisujen vaikea tulkittavuus ja niiden muodostamisessa tarvittava proseduraalinen ajattelutapa. XQueryn ilmaisuvoima perustuu osittain laajaan funktiovalikoimaan ja osittain siihen, että käyttäjä kykenee omien funktioproseduurien määrittelyyn. Nämä funktiot yhdistettynä FLWR-lauseeseen tekevät siitä tietyssä mielessä proseduraalisen ohjelmointikielen ja kyselykielen hybridin, missä menetetään molempien parhaat puolet.

Pahimmillaan XQuery-ilmaisuissa tuloksen esittämiseen ja määrittämiseen käytetyt ilmaisut liittyvät toisiinsa siten, että jonkun muun kuin kyselyn tekijän on hyvin vaikea enää ymmärtää, mikä merkitys kyselyllä on. Esimerkiksi kyselyn 5 yhteydessä kyseinen piirre ilmenee selkeästi. Siinä tuloksen tuottamiseksi on ilmaistava muuttujien viittamien elementtien alielementtien yhtäsuuruus antamalla kolme eri ilmaisua. Tutkielman kielessä ei ole lainkaan merkitystä, missä järjestyksessä nämä kolme ilmaisua on annettu. Riittää, että ne on esitetty kielen syntaksin mukaisesti ehto-osassa. Sen sijaan XQueryssä muuttujat on sijoitettava juuri oikeassa järjestyksessä for- ja let- ilmaisuihin ja niiden vertailut on annettava tarkalleen tietyissä kohdissa kyselyn kokonaisilmaisua. Ilmaisu on parhaimmillaankin virheherkkä ja saattaa tuottaa virheellisiä vastauksia käyttäjän niitä huomaamatta. Tutkielman kielessä ei jouduta tukeutumaan tällaisiin toteutusorientoituneisiin ilmaisuihin, vaan kysely voidaan ilmaista täysin deklaratiivisesti.

XQueryn ilmaisuvoimaa laajentaa mahdollisuus käyttäjän omiin funktio-määrittelyihin, missä sallitaan muun muassa rekursiivinen ohjelmointi. Kuitenkin tämä voi olla myös haitta siinä missä etukin, sillä rekursiivisen

ohjelmoinnin hallinta ei ole mitenkään itsestään selvä asia, etenkin kun kyseessä on kysely- eikä ohjelmointikieli. Kahden elementin välistä transitiivista (muuta kuin XPathin polkuaskelein ilmaistavaa) suhdetta ei XQueryssä voida esittää muulla tavoin kuin rekursiivisella funktiolla. Tutkielman kieli tarjoaa tähän käsitteellisesti merkittävästi helpomman tavan ilmaisemalla suhde kielen transitiivisen operaattorin avulla. Tämä sallii monimutkaisenkin transitiivisen suhteen ilmaisemisen kahden muuttujan välillä ilman varsinaista rekursiivista ohjelmointia.

Helppokäyttöiseksi kyselykieleksi XQuery on peruslähestymistavaltaan liian toteutusorientoitunut. Kyselyiden määrittely vaatii sellaisten seikkojen huomioon ottamista, jotka eivät oikeastaan liity lainkaan kyselyn semantiikkaan, kuten for- ja let-lauseiden keskinäinen suoritusjärjestys ja muuttujien keskinäiset suhteet iteroinnissa. Tutkielmassa kehitetty kieli tarjoaa tälle deklaratiivisen vaihtoehdon. Kielellä ilmaistaan tuloksen rakenne ja sen suhde olemassa oleviin dokumentteihin vastapainona XQueryn FLWR-lauseen algoritmiselle ajattelutavalle.

Tutkielman kieli ei sellaisenaan ole täydellinen XML-kyselykieli. Esimerkiksi se ei ota kantaa XML:n entiteettien (entity) määrittelyyn eikä elementtien nimiavaruuksiin (namespace). Entiteetit eivät ole varsinaisesti dokumentin rakennetta muodostavia osasia, vaan pikemminkin ne ovat joko yksittäisiä merkkejä tai esiprossesoinnin tuloksena dokumenttiin sulautuneita osia. Kyselykielen näkökulmasta entiteettien poisjättäminen on melko luonnollista, koska kyselykielen lähtöoletuksena on XML-muotoisen datan olemassaolo. Entiteetit ovat pikemminkin XML-dokumentin kirjoittajan työväline kuin mikään itsetarkoituksellinen tiedon piirre valmiissa dokumentissa. Samaten kyselykielen prototyyppi ei ota nimiavaruuksia huomioon, vaan katsoo XML:n nimiavaruuden tunnisteen, erottavan kaksoipisteen (:) ja lokaaliosan olevan atominen elementin nimi. Tutkielman painopisteen huomioon ottaen kieli tarjoaa ilman näitäkin piirteitä riittävän ilmaisuvoiman. Kuitenkin kielen jatkokehitetyksessä voitaisiin sekä entiteettien että nimiavaruuksien käsittelyyn tarjota jonkinlaiset ilmaisut.

Tutkielmassa kehitetty kieli voisi lisäksi tarjota myös kehittyneempiä ilmaisuja tiedon numeeriseen käsittelyyn. Esimerkiksi niiden avulla voitaisiin laskea tietynlaisten kokonaisuuksien (kuten elementtien tai tekstinpätkien) esiintymien lukumäärä jossakin dokumentissa tai elementissä. Samoin numeerista dataa sisältävien elementtien keskiarvo, maksimi ja minimi olisi kyettävä laskemaan. Tutkittava olisi ainakin, miten tällainen voitaisiin esittää ilman suurta joukkoa erilaisia ilmaisuja.

Nykyisessä muodossaan kieli tarjoaa myös tietynlaisen omien ehtoilmaisujen määrittelyn. Kuitenkin ihanteellinen ratkaisu olisi, että kielessä voitaisiin määritellä eräänlainen "lyhenneilmaisu" perustuen ehto-osan omaan syntaksiin. Tämä vaatisi kuitenkin kielen ilmaisuvoiman täydentämistä muun muassa edellä mainitulla numeerisen tiedon käsittelyominaisuuksilla.

Kuitenkin jo nykyisessä muodossaan tutkielmassa kehitetty kyselykieli tarjoaa monipuolisen, helppokäyttöisen ja deklaratiivisen tavan käsitellä XML-dokumentteja.

## Viiteluettelo

[Buneman et al, 2002] Peter Buneman, Susan Davidson, Wenfei Fan and Carmem Hara. Keys for XML. *Computer Networks* ***39***, pp 473-487.

[Clocksin&Mellish, 1984] William F. Clocksin and Christopher S. Mellish. *Programming in Prolog Second Edition*. Springer-Verlag, Berlin Heidelberg 1984.

[Elmasri&Navathe, 2000] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems Third Edition*. Addison-Wesley, Reading Massachusetts 2000.

[Fuhr&Grossjohann, 2001] Norbert Fuhr and Kai Grossjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In: *Proceedings of the 24th annual international ACM SIGIR* pp 172-180.

[ISO/IEC 14977, 1996] International standard ISO/IEC 14977 : 1996(E). Extended BNF. Draft. 1996.

[Lechner et al, 2002] Stephan Lechner , Günter Preuner and Michael Schrefl. Translating XQuery into XSLT. *Lecture Notes in Computer Science* ***2465***, pp 239-252.

[Lenz, 2002] Evan Lenz. XQuery: Reinventing the Wheel? Available as *http://www.xmlportfolio.com/xquery.html*.

[Merikoski et al, 1998] Jorma Merikoski, Ari Virtanen, Pertti Koivisto. Diskreetti Matematiikka I. Moniste, Tampereen yliopiston matemaatisten tieteiden laitos 1998.

[MSXSL] MSXSL-program and documentation. Available as http://www.msdn.microsoft.com/library/en-us/dnxslgen/html/msxsl.asp.

[Negnevitsky, 2001] Michael Negnevitsky. *Artificial Intelligence: A Guide to Intelligent Systems.* Addison-Wesley, Reading Massachusetts 2001.

[Niemi, 2001] Timo Niemi. Logiikkaohjelmoinnin luentojen oheismateriaalia 2001. Moniste, Tampereen yliopisto 2001.

[Nykänen, 2001] Ossi Nykänen. *XML*. Docendo Finland Oy, Jyväskylä 2001.

[Sterling&Shapiro, 1994] Leon Sterling and Leon Shapiro. *The Art of Prolog Second Edition*. The MIT Press, Cambridge Massachusetts 1994.

[XML, 2000] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen and Eve Maler. Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation 6 October 2000. Available as *http://www.w3.org/TR/2000/REC-xml-20001006/*.

[XMLSchema] David C. Fallside. XML Schema Part 0: Primer. W3C Recommendation, 2 May 2001. Available as *http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/*.

[XPath, 1999] James Clark, Steve DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation 16 November 1999. Available as *http://www.w3.org/TR/xpath/*.

[XPath, 2002] Anders Berglund et al. XML Path Language (XPath) 2.0. W3C Working Draft 15 November 2002. Available as *http://www.w3.org/TR/xpath20/*.

[XQuery, 2002] XQuery 1.0: An XML Query Language. W3C Working Draft 15 November 2002. Available as *http://www.w3.org/TR/xquery/*.

[XQuery op, 2002] XQuery 1.0 and XPath 2.0 Functions and Operators. W3C Working Draft 15 November 2002. Available as *http://www.w3.org/TR/xquery-operators/*.

[XQuery use, 2002] XQuery 1.0 Use Cases. W3C Working Draft 15 November 2002. Available as *http://www.w3.org/TR/xmlquery-use-cases/*.

[XSLT, 1999] James Clark. XSL Transformations (XSLT) Version 1.0 W3C Recommendation, 16 November 1999. Available as *http://www.w3.org/TR/xslt/*.

[XSLT, 2002] Michael Kay. XSL Transformations (XSLT) Version 2.0 W3C Working Draft 15 November 2002. Available as *http://www.w3.org/TR/xslt20/*.

## Dokumentti "library1.xml"

```
<bib>
  <book year="1981">
    <title>An Introduction to Database Systems</title>
    <author>
      <last>Date</last>
      <first>C</first>
    </author>
    <publisher>Addison-Wesley</publisher>
  </book>
  <book year="1993">
    <title>A Guide to the SQL Standard</title>
    <author>
      <last>Date</last>
      <first>C</first>
    </author>
    <author>
      <last>Darwen</last>
      <first>H</first>
    </author>
    <publisher>Addison-Wesley</publisher>
  </book>
  <book year="1984">
    <title>Programming in Prolog</title>
    <author>
      <last>Clocksin</last>
      <first>W</first>
    </author>
    <author>
      <last>Mellish</last>
      <first>C</first>
    </author>
    <publisher>Springer-Verlag</publisher>
  </book>
  <book year="1992">
    <title>Deductive Databases and Logic Programming</title>
    <author>
      <last>Das</last>
      <first>S</first>
    </author>
    <publisher>Addison-Wesley</publisher>
  </book>
  <book year="1996">
    <title>Multimedia Database Systems</title>
    <editor>
      <last>Subramanian</last>
      <first>S</first>
    </editor>
    <editor>
      <last>Jajodia</last>
      <first>S</first>
    </editor>
    <publisher>Springer-Verlag</publisher>
  </book>
  <book year="1972">
    <title>Algorithms + Data Structures = Programs</title>
    <author>
      <last>Wirth</last>
      <first>N</first>
    </author>
    <publisher>Prentice-Hall</publisher>
  </book>
</bib>
```

## Dokumentti "library2.xml":

```
<entries>
  <entry>
    <title>An Introduction to Database Systems</title>
    <pub>
      <name>Addison-Wesley</name>
      <year>1981</year>
    </pub>
    <class>DB</class>
  </entry>
  <entry>
    <title>Programming in Prolog</title>
    <pub>
      <name>Springer-Verlag</name>
      <year>1984</year>
    </pub>
    <class>LP</class>
  </entry>
  <entry>
    <title>Deductive Databases and Logic Programming</title>
    <pub>
      <name>Addison-Wesley</name>
      <year>1992</year>
    </pub>
    <class>DB</class>
    <class>LP</class>
  </entry>
  <entry>
    <title>The Art of Prolog</title>
    <pub>
      <name>MIT Press</name>
      <year>1994</year>
    </pub>
    <class>LP</class>
  </entry>
  <entry>
    <title>Fundamentals of Database Systems</title>
    <pub>
      <name>Addison-Wesley</name>
      <year>2000</year>
    </pub>
    <class>DB</class>
  </entry>
</entries>
```

## Dokumentti "student.xml":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE students [
  <!ELEMENT students (student*) >
  <!ELEMENT student (name,id,start_year)>
  <!ELEMENT name  (#PCDATA )>
  <!ELEMENT id  (#PCDATA )>
  <!ELEMENT start_year  (#PCDATA )>
] >
<students>
  <student>
    <name>Robert Poster</name>
    <id>121</id>
    <start_year>1999</start_year>
  </student>
  <student>
    <name>Joe Jackson</name>
    <id>111</id>
    <start_year>2001</start_year>
  </student>
  <student>
    <name>Mark Mills</name>
    <id>120</id>
    <start_year>1999</start_year>
  </student>
  <student>
    <name>Paul Jones</name>
    <id>266</id>
    <start_year>2000</start_year>
  </student>
  <student>
    <name>Peter Gilardi</name>
    <id>155</id>
    <start_year>2001</start_year>
  </student>
  <student>
    <name>Ephraim Katz</name>
    <id>265</id>
    <start_year>2000</start_year>
  </student>
  <student>
    <name>Pavel Stransky</name>
    <id>366</id>
    <start_year>2000</start_year>
  </student>
</students>
```

## Dokumentti "teacher.xml"

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE teachers [
  <!ELEMENT teachers (teacher*) >
  <!ELEMENT teacher (name,id)>
  <!ELEMENT name  (#PCDATA )>
  <!ELEMENT id  (#PCDATA )>
] >
<teachers>
  <teacher>
    <name>Gibson</name>
    <id>GIB</id>
  </teacher>
  <teacher>
    <name>Jones</name>
    <id>JON</id>
  </teacher>
  <teacher>
    <name>Moore</name>
    <id>MOO</id>
  </teacher>
  <teacher>
    <name>Mills</name>
    <id>MIL</id>
  </teacher>
</teachers>
```

Dokumentti "course.xml"

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE courses [
  <!ELEMENT courses (course*) >
  <!ELEMENT course (name,preq?,id)>
  <!ELEMENT name  (#PCDATA )>
  <!ELEMENT preq (id+)>
  <!ELEMENT id  (#PCDATA )>
] >
<courses>
  <course>
    <name>Introduction to Programming</name>
    <id>CS1</id>
  </course>
  <course>
    <name>Data Structures</name>
    <id>CS2</id>
  </course>
  <course>
    <name>Database Programming</name>
    <id>CS3</id>
    <preq>
      <id>CS1</id>
      <id>CS2</id>
    </preq>
  </course>
  <course>
    <name>Logic Programming</name>
    <id>CS4</id>
    <preq>
      <id>CS2</id>
    </preq>
  </course>
</courses>
```

## Dokumentti "lecture.xml"

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE lectures [
  <!ELEMENT lectures (lecture*) >
  <!ELEMENT lecture (course,season,teacher,grades)>
  <!ELEMENT course (#PCDATA )>
  <!ELEMENT season (year,term)>
  <!ELEMENT year (#PCDATA )>
  <!ELEMENT term (#PCDATA )>
  <!ELEMENT teacher  (#PCDATA )>
  <!ELEMENT grades (grade*) >
  <!ELEMENT grade (student,mark) >
  <!ELEMENT student (#PCDATA) >
  <!ELEMENT mark (#PCDATA) >
] >
<lectures>
  <lecture>
    <course>CS1</course>
    <season>
      <year>1999</year>
      <term>A</term>
    </season>
   <teacher>GIB</teacher>
   <grades>
     <grade><student>120</student><mark>2</mark></grade>
     <grade><student>121</student><mark>1</mark></grade>
   </grades>
  </lecture>
  <lecture>
    <course>CS1</course>
    <season>
      <year>2000</year>
      <term>A</term>
    </season>
   <teacher>GIB</teacher>
   <grades>
     <grade><student>265</student><mark>2</mark></grade>
     <grade><student>266</student><mark>1</mark></grade>
   </grades>
  </lecture>
  <lecture>
    <course>CS1</course>
    <season>
      <year>2001</year>
      <term>A</term>
    </season>
   <teacher>MOO</teacher>
   <grades>
     <grade><student>111</student><mark>2</mark></grade>
     <grade><student>155</student><mark>3</mark></grade>
     <grade><student>366</student><mark>2</mark></grade>
   </grades>
  </lecture>
  <lecture>
    <course>CS2</course>
    <season>
      <year>2000</year>
      <term>S</term>
    </season>
   <teacher>JON</teacher>
   <grades>
     <grade><student>120</student><mark>3</mark></grade>
```

```
   <grade><student>121</student><mark>1</mark></grade>
 </grades>
</lecture>
<lecture>
  <course>CS2</course>
  <season>
    <year>2001</year>
    <term>S</term>
  </season>
 <teacher>JON</teacher>
 <grades>
   <grade><student>265</student><mark>2</mark></grade>
   <grade><student>266</student><mark>1</mark></grade>
 </grades>
</lecture>
<lecture>
  <course>CS2</course>
  <season>
    <year>2002</year>
    <term>S</term>
  </season>
 <teacher>JON</teacher>
 <grades>
   <grade><student>111</student><mark>2</mark></grade>
   <grade><student>155</student><mark>1</mark></grade>
   <grade><student>366</student><mark>1</mark></grade>
 </grades>
</lecture>
<lecture>
  <course>CS3</course>
  <season>
    <year>1999</year>
    <term>A</term>
  </season>
 <teacher>MOO</teacher>
 <grades>
   <grade><student>120</student><mark>3</mark></grade>
 </grades>
</lecture>
<lecture>
  <course>CS3</course>
  <season>
    <year>2001</year>
    <term>A</term>
  </season>
 <teacher>GIB</teacher>
 <grades>
   <grade><student>121</student><mark>1</mark></grade>
   <grade><student>155</student><mark>1</mark></grade>
   <grade><student>265</student><mark>3</mark></grade>
   <grade><student>266</student><mark>2</mark></grade>
 </grades>
</lecture>
<lecture>
  <course>CS4</course>
  <season>
    <year>2001</year>
    <term>S</term>
  </season>
 <teacher>MIL</teacher>
 <grades>
   <grade><student>120</student><mark>2</mark></grade>
   <grade><student>266</student><mark>3</mark></grade>
 </grades>
</lecture>
<lecture>
  <course>CS4</course>
```

```
    <season>
      <year>2002</year>
      <term>S</term>
    </season>
   <teacher>MIL</teacher>
   <grades>
     <grade><student>121</student><mark>1</mark></grade>
     <grade><student>265</student><mark>3</mark></grade>
     <grade><student>366</student><mark>2</mark></grade>
   </grades>
  </lecture>
</lectures>
```

## Dokumentti "bicycle.xml"

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <bicycle>
  <part>
   <name>bicycle mk0</name>
   <part_id>0</part_id>
   <prod_started>1980</prod_started>
   <prod_ended>2002</prod_ended>
  </part>
  <part>
   <name>bicycle mk1</name>
   <part_id>1</part_id>
   <prod_started>1995</prod_started>
   <prod_ended>2000</prod_ended>
  </part>
  <part>
   <name>bicycle mk2</name>
   <part_id>2</part_id>
   <prod_started>1998</prod_started>
  </part>
  <part>
   <name>bicycle mk3</name>
   <part_id>3</part_id>
   <prod_started>2002</prod_started>
  </part>
  <part>
   <name>front_wheel mk1</name>
   <part_id>41</part_id>
   <prod_started>1995</prod_started>
   <part_of>0</part_of>
   <part_of>1</part_of>
   <part_of>2</part_of>
  </part>
  <part>
   <name>front_wheel mk2</name>
   <part_id>42</part_id>
   <prod_started>2002</prod_started>
   <part_of>3</part_of>
  </part>
  <part>
   <name>rear_wheel mk1</name>
   <part_id>5</part_id>
   <prod_started>1995</prod_started>
   <part_of>0</part_of>
  </part>
  <part>
   <name>rear_wheel mk2</name>
   <part_id>6</part_id>
   <prod_started>1998</prod_started>
   <part_of>1</part_of>
   <part_of>2</part_of>
  </part>
  <part>
   <name>rear_wheel mk3</name>
   <part_id>7</part_id>
   <prod_started>2002</prod_started>
   <part_of>3</part_of>
  </part>
  <part>
   <name>body mk1</name>
   <part_id>8</part_id>
   <prod_started>1995</prod_started>
   <part_of>0</part_of>
   <part_of>1</part_of>
```

```
</part>
<part>
 <name>body mk2</name>
 <part_id>9</part_id>
 <prod_started>1998></prod_started>
 <part_of>2</part_of>
</part>
<part>
 <name>body mk3</name>
 <part_id>8</part_id>
 <prod_started>1995></prod_started>
 <part_of>3</part_of>
</part>
<part>
 <name>handlebar mk1</name>
 <part_id>10</part_id>
 <prod_started>1995</prod_started>
 <part_of>0</part_of>
 <part_of>1</part_of>
</part>
<part>
 <name>handlebar mk2</name>
 <part_id>11</part_id>
 <prod_started>1998</prod_started>
 <part_of>2</part_of>
</part>
<part>
 <name>handlebar mk3</name>
 <part_id>12</part_id>
 <prod_started>2002</prod_started>
 <part_of>3</part_of>
</part>
<part>
 <name>bell</name>
 <part_id>13</part_id>
 <prod_started>1995</prod_started>
 <prod_ended>1999</prod_ended>
 <part_of>10</part_of>
 <part_of>11</part_of>
</part>
<part>
 <name>light</name>
 <part_id>14</part_id>
 <prod_started>2002</prod_started>
 <part_of>11</part_of>
 <part_of>12</part_of>
</part>
<part>
 <name>single speed gear</name>
 <part_id>15</part_id>
 <prod_started>1995</prod_started>
 <part_of>5</part_of>
</part>
<part>
 <name>3-speed gear</name>
 <part_id>16</part_id>
 <prod_started>1995</prod_started>
 <part_of>6</part_of>
</part>
<part>
 <name>6-speed gear</name>
 <part_id>17</part_id>
 <prod_started>2002</prod_started>
 <part_of>7</part_of>
</part>
<part>
 <name>saddle mk1</name>
 <part_id>18</part_id>
 <prod_started>1995</prod_started>
 <part_of>0</part_of>
 <part_of>1</part_of>
</part>
<part>
 <name>saddle mk2</name>
 <part_id>19</part_id>
 <prod_started>1998</prod_started>
 <part_of>2</part_of>
 <part_of>3</part_of>
</part>
<part>
 <name>pedals mk1</name>
 <part_id>20</part_id>
 <prod_started>1995</prod_started>
```

```
   <part_of>8</part_of>
  </part>
  <part>
   <name>pedals mk2</name>
   <part_id>21</part_id>
   <prod_started>1998</prod_started>
   <part_of>9</part_of>
  </part>
  <part>
   <name>steel rim</name>
   <part_id>22</part_id>
   <prod_started>1980</prod_started>
   <part_of>5</part_of>
   <part_of>6</part_of>
   <part_of>41</part_of>
  </part>
  <part>
   <name>coal fiber rim</name>
   <part_id>23</part_id>
   <prod_started>2002</prod_started>
   <part_of>7</part_of>
   <part_of>42</part_of>
  </part>
 </bicycle>
```

## Dokumentin "art_001.xml" DTD:

```
<!ELEMENT doc (frontmatter,docbody,referencelist?,appendix*)>
<!ELEMENT frontmatter
(title,authorlist,jname?,pyear,vol-issue?,pages?,publisher,abstract?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT authorlist (author+)>
<!ELEMENT jname (#PCDATA)>
<!ELEMENT pyear (#PCDATA)>
<!ELEMENT vol-issue (#PCDATA)>
<!ELEMENT pages (#PCDATA)>
<!ELEMENT publisher (pname,paddress?)>
<!ELEMENT pname (#PCDATA)>
<!ELEMENT paddress (#PCDATA)>
<!ELEMENT author (fn,sn,affiliation?)>
<!ELEMENT fn (#PCDATA)>
<!ELEMENT sn (#PCDATA)>
<!ELEMENT affiliation (#PCDATA)>
<!ELEMENT abstract (#PCDATA)>
<!ELEMENT docbody (chapter+)>
<!ELEMENT chapter (cheading,(text|section)+)>
<!ELEMENT cheading (#PCDATA)>
<!ELEMENT section (sheading,text)>
<!ELEMENT sheading (#PCDATA)>
<!ELEMENT text (paragraf|table|fig)+>
<!ELEMENT paragraf (#PCDATA|rid)*>
<!ELEMENT rid (#PCDATA)>
<!ELEMENT table (tcaption,tbody)>
<!ELEMENT fig (fcaption,fbody)>
<!ELEMENT tcaption (#PCDATA)>
<!ELEMENT tbody (#PCDATA)>
<!ELEMENT fcaption (#PCDATA)>
<!ELEMENT fbody (#PCDATA)>
<!ELEMENT referencelist (ref+)>
<!ELEMENT ref (rid,authorlist,ryear,rtitle,rsource)>
<!ELEMENT ryear (#PCDATA)>
<!ELEMENT rtitle (#PCDATA)>
<!ELEMENT rsource (#PCDATA)>
<!ELEMENT appendix ANY>
```

## Dokumentti "art_001.xml"

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE doc SYSTEM "doc.dtd">
<doc>
  <frontmatter>
    <title>Keys for XML</title>
    <authorlist>
      <author><fn>Peter</fn><sn>Buneman</sn><affiliation>University of
Pennsylvania</affiliation></author>
      <author><fn>Susan</fn><sn>Davidson</sn><affiliation>University
of Pennsylvania</affiliation></author>
      <author><fn>Wenfei</fn><sn>Fan</sn><affiliation>Bell
Labs</affiliation></author>
      <author><fn>Carmem</fn><sn>Hara</sn><affiliation>Universidade
Federal do Parana</affiliation></author>
    </authorlist>
    <jname>Computer Networks</jname>
    <pyear>2002</pyear>
    <vol_issue>39</vol_issue>
    <pages>473–487</pages>
    <publisher>
      <pname>Elsevier Science
B.V.</pname><paddress>www.elsevier.com/locate/comnet</paddress>
    </publisher>
    <abstract>We discuss the definition of keys for
XML documents, paying particular attention to the
concept of a relative key,
which is commonly used in hierarchically structured
documents and scientific databases.</abstract>
   </frontmatter>
   <docbody>
     <chapter>
      <cheading>1. Introduction</cheading>
      <text>
        <paragraph>Keys are an essential part of database design
<rid>2</rid><rid>19</rid>: they are fundamental to data models and
conceptual design...</paragraph>
      </text>
      <section>
        <sheading>1.1. Organization</sheading>
        <text>
          <paragraph>The rest of the paper is organized as follows.
Section 2 introduces the notion of node addresses
and value equality. Node addresses are used in
node equality testing...</paragraph>
        </text>
      </section>
    </chapter>
    <chapter>
      <cheading>2. Node addresses and equality</cheading>
      <text>
        <paragraph>The document object model (DOM) provides
some insight into a semantics for XML
documents. According to the DOM, a document is
a hierarchical structure of nodes. Nodes are of
several types, but there are three types that are
important to this discussion: element nodes, attribute
nodes, and text nodes. As illustrated in Fig.
1 text nodes (T) have no name but carry text, attribute
nodes (A) have both a name and carry text,
and element nodes (E) have a name. Element
nodes may have children...</paragraph>
<fig><fcaption>Fig. 1. Some XML and its representation as a
tree.</fcaption><fbody>http://url.figure1.fig</fbody></fig>
        <paragraph>A consequence of this model is that a path of
```

```
edge labels from the root uniquely identifies a
node. We shall call such paths node addresses...</paragraph>
     </text>
     </chapter>
     <chapter>
       <cheading>3. Path expressions</cheading>
       <text>A path expression is an expression involving
node names (tags and attribute names) that describes
a set of paths in the document tree.
The choice of what language we use to define
path expressions is important to the expressive
power of keys, and there are a number of choices.
In XML-Schema, XPath <rid>15</rid> expressions...</text>
     </chapter>
  </docbody>
<referencelist>
  <ref>
    <rid>2</rid>
    <authorlist>
      <author><fn>S.</fn><sn>Abiteboul</sn></author>
      <author><fn>R.</fn><sn>Hull</sn></author>
      <author><fn>V.</fn><sn>Vianu</sn></author>
    </authorlist>
    <ryear>1995</ryear>
    <rtitle>Foundations of Databases</rtitle>
    <rsource>Addison-Wesley</rsource>
  </ref>
  <ref>
    <rid>15</rid>
    <authorlist>
      <author><fn>J.</fn><sn>Clark</sn></author>
      <author><fn>S.</fn><sn>DeRose</sn></author>
    </authorlist>
    <ryear>1999</ryear>
    <rtitle>XML Path Language (XPath)</rtitle>
    <rsource>http://www.w3.org/TR/xpath</rsource>
  </ref>
   <ref>
    <rid>19</rid>
    <authorlist>
      <author><fn>R.</fn><sn>Ramakrishnan</sn></author>
      <author><fn>J.</fn><sn>Gehrke</sn></author>
    </authorlist>
    <ryear>2000</ryear>
    <rtitle>Database Management Systems</rtitle>
    <rsource>McGraw-Hill Higher Education</rsource>
  </ref>
</referencelist>
<appendix>
</appendix>
</doc>
```

## Kielen syntaksi, BNF-notaatio

query ::=

"get" result_element

"with variables" variables [ "with condition" condition ] "." ;

result_element ::=

element_name "(" result_token { "," result _token } ")" ;

result_token ::= variable_path | text | user_element | group_element;

variable_path ::= variable [ path_step ] ;

path_step ::=

path_separator

( path_operator | (element_name [position] | ". ." | "*" | alternate_names)

[ path_step ] ) ;

path_separator ::= "/" | "//" ;

path_operator ::= "#" [ "name" | "doc" | "path"] ;

position ::= "[" ( ("last" ["-" number ] ) | number ) "]";

alternate_names ::= "(" element_name { "|" element_name } ")";

user_element ::= element_name "(" result_token { "," result _token } ")" ;

group_element ::= element_name "{" result_token { "," result _token } "}" ;

variables ::= variable_definition { "," variable_definition} ;

variable_definition ::= variable { "," variable } origin ;

origin ::= "from" "(" document_definition { "," document_definition } ")" ;

document_definition ::= source_doc [ search_path | root_path];

source_doc ::= ("any" | doc_name | "query" "(" query ")" );

search_path ::= path_separator search_name { search_path }.

search_name ::= (element_name | "*")[position].

root_path ::= "/".

condition ::= boolean_expression ;

boolean_expression ::= "(" logical_factor { "or" logical_factor } ")" ;

logical_factor ::= ["not"] logical_element { "and" logical_factor};

logical_element ::= primitive | boolean_expression;

primitive ::= primitive1 | primitive2 | primitive3 | primitive4 | primitive5 |
primitive6 | primitive7 | primitive8 | primitive9 | trans_op

```
primitive1 ::=
        variable ("has" | "contains") "(" variable_path { "," variable_path} ")" ;
primitive2 ::=
       variable_path ("precedes" | "before") variable_path ;
primitive3 ::=
      variable_path ("has" | "contains") "elements"
         "(" quantity_token {"," quantity_token}")";
primitive4 ::=
      variable_path "is" ("empty" | "root" | ("type one of" name_list )) ;
primitive5 ::= variable_path ("=" | "!=") variable_path | value ;
primitive6 ::= variable_path (">" | "<") variable_path | number ;
primitive7 ::= variable_path ">>=" variable_path ;
primitive8 ::=
            variable_path ("contains" | "has") "text as"
            ("in" variable_path) | ("one of" variable_path_list) ;
primitive9 ::= variable_path ( "has" | "contains" ) "text" pattern_list ;

trans_op ::= "transitive" variable "to" variable boolean_expression;

pattern_list ::= "(" pattern_token { "or" pattern_token } ")" ;
pattern_token ::= ["not"] pattern_element { "and" pattern_token } ;
pattern_element ::= pattern_text | pattern_list | one_of_textlist | near_list |
sim_list ;
pattern_text ::= qt {alphanum | misc | "*" | "?"} qt;
one_of_textlist ::= "one of" "(" pattern_text {"," pattern_text }")" ;
near_list ::= "near" number "for" "(" pattern_text {"," pattern_text }")" ;
sim_list ::=
      [hedge] "sim" "(" pattern_text {"," pattern_text }")" [membership];
hedge ::= "slightly" | "very" | "extremely" | "somewhat" | "indeed" ;
membership ::= number;

quantity_token ::= element_name ["<" | ">"] number;
name_list ::= "(" element_name {"," element_name} ")".
value ::= text | number ;
variable_path_list ::= variable_path {"," variable_path};
variable ::= lower_alpha{alphanum};
alphanum ::= number | alpha;
number ::= ("0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9") {number};
alpha ::= lower_alpha | upper_alpha;
```

```
element_name::= lower_alpha{alphanum};
text ::= qt {alphanum | misc} qt;
lower_alpha ::= ("a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" |
                 "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" |
                 "u" | "v" | "w" | "x" | "y" | "z");
upper_alpha ::= ("A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" |
                 "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" |
                 "U" | "V" | "W" | "X" | "Y" | "Z");
```

## DCG-esimerkit

## Lukusanoja jäsentävä ohjelma

```
number( 0, en )-->"zero".
number( 0, fi )-->"nolla".
number( N, Lang )-->xxx(N, Lang).

xxx(N,en)-->digit(D,en)," hundred",rest_xxx(N1,en), {N is D*100+N1}.
xxx(N,Lang)-->xx(N,Lang).

xxx(N,fi)-->
        digit(D,fi),{D > 1},"sataa",rest_xxx(N1,fi), {N is D*100+N1}.
xxx(N,fi)-->"sata",rest_xxx(N1,fi), {N is 100+N1}.

rest_xxx(0,Lang)-->[].
rest_xxx(N,en)-->" and ",xx(N,en). rest_xxx(N,fi)-->xx(N,fi).

xx(D, Lang)-->digit(D,Lang).
xx(T, Lang)-->teen(T,Lang).
xx(N, Lang)-->tens(T,Lang), rest_xx(N1,Lang), {N is T + N1}.

rest_xx(0,Lang)-->[].
rest_xx(N,fi)-->digit(N,fi).
rest_xx(N,en)-->" ",digit(N,en).

digit(1,en)-->"one".    digit(1,fi)-->"yksi".
digit(2,en)-->"two".    digit(2,fi)-->"kaksi".
digit(3,en)-->"three".  digit(3,fi)-->"kolme".
digit(4,en)-->"four".   digit(4,fi)-->"nelja".
digit(5,en)-->"five".   digit(5,fi)-->"viisi".
digit(6,en)-->"six".    digit(6,fi)-->"kuusi".
digit(7,en)-->"seven".  digit(7,fi)-->"seitseman".
digit(8,en)-->"eight".  digit(8,fi)-->"kahdeksan".
digit(9,en)-->"nine".   digit(9,fi)-->"yhdeksan".

teen(10,en)-->"ten".            teen(11,en)-->"eleven".
teen(12,en)-->"twelve".         teen(13,en)-->"thirteen".
teen(14,en)-->"fourteen".       teen(15,en)-->"fifteen".
teen(16,en)-->"sixteen".        teen(17,en)-->"seventeen".
teen(18,en)-->"eighteen".       teen(19,en)-->"nineteen".

teen(10,fi)-->"kymmenen".
teen(N,fi)-->digit(N1,fi),"toista",{N is 10 + N1}.

tens(20,en)-->"twenty".         tens(30,en)-->"thrirty".
tens(40,en)-->"forty".          tens(50,en)-->"fifty".
tens(60,en)-->"sixty".          tens(70,en)-->"seventy".
tens(80,en)-->"eighty".         tens(90,en)-->"ninety".

tens(N,fi)-->digit(N1,fi),{ N1 > 1 },"kymmenta",{N is 10 * N1}.

en_fi_number(Number,Numero,X):-
        number( X, fi, Numero, [] ),
        number( X, en, Number, [] ),
        name(Nimi,Numero),
        name(Name,Number),
        write('EN: '),write(Name),
        write(' - FI: '),write(Nimi),nl.
```

## Päättymättömän rekursion esto Iterative Deepening Search-menetelmällä

```
s( N ) --> a( NA ), b( NB ), c( NC ), { N is NA + NB + NC }.
a( N ) --> [a], a( N1 ),{ N is N1 +1 }.
a( 0 ) -->[].
b( N ) --> [b], b( N1 ),{ N is N1 +1 }.
b( 0 ) -->[].
c( N ) --> [c], c( N1 ),{ N is N1 +1 }.
c( 0 ) -->[].


pattern_list([]).
pattern_list([X|Xs]):- pattern_list(Xs).
```

Esimerkkiajo LPA-Prologissa:

```
| ?- s(N,S,[]).
Error 4, Heap Space Full, Trying a/3

Aborted
| ?- pattern_list(S),s(N,S,[]).
S = [] ,
N = 0 ;

S = [a] ,
N = 1 ;

S = [b] ,
N = 1 ;

S = [c] ,
N = 1 ;

S = [a,a] ,
N = 2 ;

S = [a,b] ,
N = 2 ;

S = [a,c] ,
N = 2 ;
```

*etc.*

Listan pituus määräytyy ennakolta, joten hakupuu täyttyy taso kerrallaan.

## Yksinkertainen XML-kielioppi

```
xml(X)-->prolog,xmlelement(X),chars.

prolog-->s(*).

xmlelement(element(X,As,[]))-->
        "<",name(X),attributes(As),s(*),"/>".
xmlelement(element(X,As,Es))-->
        "<",name(X),attributes(As),s(*),">",
        content(Es),
        "</",name(X),">".

name(X)--> namechars(C),{name(X,C)}.

namechars([C])-->namechar(C).
namechars([C|Cs])-->namechar(C), namechars(Cs).

content([])-->s(*).
content([X|Xs])-->s(*),xmlelement(X),content(Xs).
content([X|Xs])-->pcdata_string(X),con_element(Xs).

con_element( [] )-->[].
con_element( [X|Xs] )-->xmlelement(X),content(Xs).

attributes([])-->[].
attributes([A|As])-->s(+),attribute(A),attributes(As),
        {not same (A,As)}.

same(A,[X|Xs]):- functor(A,Xx,_), functor(X,Xx,_),!.
same(A,[X|Xs]):- same(A,Xs).

attribute(A)-->
        name(X),s(*),eq,s(*),qt,non_qtchars(S),qt,
        { name(Av,S),A =.. [X,Av] }.

pcdata_string(pcdata(X))-->
        s(*),non_ltchars(S),s(*),{non_empty(S),string_chars(X,S)}.
/*HUOM! Tässä käytetään LPA-Prologin string-tietotyyppiä lähinnä
element-termin luettavuuden vuoksi. (Normaalilistaesitys näkyy
numerokoodeina ja atom-tietotyyppi voi olla vain 256 merkkiä pitkä)*/

non_empty([S|_]):-S > 32.
non_empty([S|Ss]):-non_empty(Ss).
qt-->[34].
eq-->"=".
namechar(C)-->[C],{ (C>=65,C=<90);(C>=97,C=<122) }.
chars-->[].
chars-->[C],chars.
/*non_qtchars ei saa sisältää lainausmerkkiä*/
non_qtchars([])-->[].
non_qtchars([C|Cs])-->[C],{ C \= 34 },non_qtchars(Cs).

/*non_ltchars ei saa sisältää pienempi kuin -merkkiä*/
non_ltchars([C|Cs])-->[C],{ C \= 60 },non_ltchars(Cs).
non_ltchars([C])-->[C],{ C \= 60 }.
s(+)-->[C],{C<33}.              s(*)-->[].
s(+)-->[C],{C<33},s(+).         s(*)-->[C],{C<33},s(*).
```

## XML-dokumentin lisäkäsittelyvälineitä

```
/*Prolog-elementtiesityksen puutulostaja*/
pr_tree(element(X,A,E),N):-
        tab(N),write(X),nl,pr_attrs(A,N),pr_children(E,N+2).

pr_attrs([],_).
pr_attrs([A|As],N):-
     tab(N),functor(A,An,_),write('@'),write(An),nl,pr_attrs(As,N).

pr_children([],_).
pr_children([element(X,A,E)|Es],N):-
        pr_tree(element(X,A,E),N),pr_children(Es,N).
pr_children([pcdata(_)|Es],N):-
        tab(N),write('#PCDATA'),nl,pr_children(Es,N).

/* Erotuslistan avulla toimiva
   tiedostonlukuohjelma
*/
read_file(Z,Chars):-
        see(Z),get0(P),do_read(P,X-X, Chars),close(Z).
do_read( -1, Xs-Zs, Xs ):-!.
do_read( P, Xs-[P|Zs], F ):-
        get0(P1),do_read2(P1,Xs-Zs,F).


/* yhdessä xml-kieliopin kanssa ajettava (tarvitaan myös jonkinlainen
XML-dokumentti, joka tässä on nimeltään doc.xml):

read_file('doc.xml',XML_As_CharArray),
        xml(X, XML_As_CharArray ,[]),
        pr_tree(X,0).
*/
```

## Kyselykielen DCG-määrittely

```
query(q_selection(Selection,Variables,Conditions))-->
   [get],selection(Selection),variables(Variables),condition(Variables,Conditions).

selection(selection(Res,[S|Ss]))-->
   [Res,'('],selected_element(S),more_selected_elements(Ss),[')'].
selection(m_sel(Res,[S|Ss]))-->
   [Res,'['],selected_element(S),more_selected_elements(Ss),[']'].
selection(m_sel(Res,[S|Ss]))-->
   [Res,'{'],selected_element(S),more_selected_elements(Ss),['}'].
selection(plain_selection(S))-->
   ['('],selected_element(S),[')'].

selected_element((Var,p(Path)))-->[Var],selection_path(Path).
selected_element(pcdata(S))-->string_as_chars(S).
selected_element(empty(S))-->[S,'(',')'].
selected_element(S)-->selection(S).

selection_path([])-->[].
selection_path([rest])-->[/#].
selection_path([name])-->[/#,name].
selection_path([doc])-->[/#,doc].
selection_path([parent|Path])-->[/..],selection_path(Path).
selection_path([ancestor|Path])-->[//..],selection_path(Path).
selection_path([hp(X,Pos)|Path])-->
  [/],id_token_name(X),position_pred(Pos),selection_path(Path).
selection_path([cp(X,Pos)|Path])-->
  [//],id_token_name(X),position_pred(Pos),selection_path(Path).
selection_path([h(X)|Path])-->[/],id_token_name(X),selection_path(Path).
selection_path([c(X)|Path])-->[//],id_token_name(X),selection_path(Path).

string_as_chars(S)-->[C],{chars(C),string_chars( S, C)}.

position_pred(Number)-->['[',Number],{number(Number)},[']'].
position_pred(last-C)-->['[',last],decrement(C),[']'].

decrement(0)-->[].
decrement(Number)-->[-,Number],{number(Number)}.

%id_token_name(X)-->[*],!.
id_token_name(X)-->[X].
id_token_name(alt([X|Xs]))-->['(',X],other_id_token_names(Xs),[')'].

other_id_token_names([])-->[].
other_id_token_names([X|Xs])-->['|',X],other_id_token_names(Xs).


more_selected_elements([])-->[].
more_selected_elements([S|Ss])-->[','],selected_element(S),more_selected_elements(Ss).

variables(S)-->[with,variables],token_list(S).

token_list(List)-->token(S),more_tokens(Ss),{append(S,Ss,List)}.

token(S)--
>new_variable(V),more_new_variables(Vs),origin(Docs),{build_tokens([V|Vs],Docs,S)}.

build_tokens([],_,[]).
build_tokens([V|Vs],Docs,[(V,_,Docs)|Rs]):-build_tokens(Vs,Docs,Rs).

more_tokens([])-->[].
more_tokens(List)-->[','],token(S),more_tokens(Ss),{append(S,Ss,List)}.


more_new_variables([])-->[].
more_new_variables([V|Vs])-->[','],new_variable(V),more_new_variables(Vs).

origin([(0,0)])-->[from,'(',any,')'],{!}.
origin([(Doc,0)|Docs])-->[from,'('],docname(Doc),more_docnames(Docs),[')'].
origin([(Doc,X)|Docs])-->[from,'('],docname(Doc),['/',X],more_docnames(Docs),[')'].



more_docnames([])-->[].
more_docnames([(Doc,0)|Docs])-->[','],docname(Doc),more_docnames(Docs).
more_docnames([(Doc,X)|Docs])-->[','],docname(Doc),selection_path(X),
                                more_docnames(Docs).
```

```
% b from "bib.xml"/book tarkoittaa ett„ b koskee book - elementti„, joka l”ytyy
"bib.xml" -dokumentista.

docname(query(Nested))-->[query,'('],query(Nested),[')'],{!}.
docname(0)-->[any],{!}.
docname(Doc)-->[D],{chars(D),name(Doc,D)}.



new_variable(X)-->[X].

condition(_,true)-->[].
condition(S,C)-->[with,condition],boolean_condition(S,C).

boolean_condition(S,Cor)-->['('],log_factor(S,C),or_log_factor(S,C,Cor),[')'].
or_log_factor(_,C,C)-->[].
or_log_factor(S,Cprv,Cor)-->[or],log_factor(S,C),or_log_factor(S,(Cprv;C),Cor).
log_factor(S,(C,Cand))-->log_elem(S,C),and_log_factor(S,Cand).
log_factor(S,((not C),Cand))-->[not],log_elem(S,C),and_log_factor(S,Cand).
and_log_factor(_,true)-->[].
and_log_factor(S,(C,Cand))-->[and],log_factor(S,C),and_log_factor(S,Cand).
log_elem(S,C)-->b_prim(S,C);boolean_condition(S,C).
b_prim(S,C)-->condition_property(S,C).

condition_property(S,elem_contains_text_as_in(T1,Path1,T2,Path2))-->
  condition_token(T1,S),condition_token_path(Path1),
  [contains,text,as,in],condition_token(T2,S),condition_token_path(Path2).
condition_property(S,elem_has_text_as_in(T1,Path1,T2,Path2))-->
  condition_token(T1,S),condition_token_path(Path1),
  [has,text,as,in],condition_token(T2,S),condition_token_path(Path2).
condition_property(S,elem_has_text_as_one_of(T1,Path1,TPath2))-->
  condition_token(T1,S),condition_token_path(Path1),
  [has,text,as,one,of],condition_token_path_list(TPath2,S).

condition_property(S,elem_has_elems(T1,T2))-->
   condition_token(T1,S),[has],condition_token_path_list(T2,S).
condition_property(S,elem_contains_elems(T1,T2))-->
   condition_token(T1,S),[contains],condition_token_list(T2,S).
condition_property(S,elem_has_elem(T2,T1))-->
   condition_token(T1,S),[is,part,of],condition_token(T2,S).
condition_property(S,elem_is_empty(T1))-->condition_token(T1,S),[is,empty].
condition_property(S,elem_is_root(T1))-->condition_token(T1,S),[is,root].

condition_property(S,elems_equal_depth(Ls))-->[equal,depth],condition_token_list(Ls,S).
condition_property(S,in_path_value_equals(T,Path,D))-->
  condition_token(T,S),condition_token_path(Path),[=],element_value(D).
condition_property(S,in_path_value_greater_than(T,Path,D))-->
  condition_token(T,S),condition_token_path(Path),[>,D].
condition_property(S,in_path_value_less_than(T,Path,D))-->
  condition_token(T,S),condition_token_path(Path),[<,D].
condition_property(S,in_path_value_inequals(T,Path,D))-->
  condition_token(T,S),condition_token_path(Path),['!',=,D].
condition_property(S,in_path_value_inequals_in_path(T1,Path1,T2,Path2))-->
  condition_token(T1,S),condition_token_path(Path1),['!',=],
  condition_token(T2,S),condition_token_path(Path2).
condition_property(S,in_path_value_equals_in_path(T1,Path1,T2,Path2))-->
  condition_token(T1,S),condition_token_path(Path1),
  [=],condition_token(T2,S),condition_token_path(Path2).
condition_property(S,in_path_value_transequals_in_path(T1,Path1,T2,Path2))-->
  condition_token(T1,S),condition_token_path(Path1),
  [>>=],condition_token(T2,S),condition_token_path(Path2).
condition_property(S,in_path_value_greater_than_in_path(T1,Path1,T2,Path2))-->
  condition_token(T1,S),condition_token_path(Path1),
  [>],condition_token(T2,S),condition_token_path(Path2).
condition_property(S,in_path_value_less_than_in_path(T1,Path1,T2,Path2))-->
  condition_token(T1,S),condition_token_path(Path1),[<],
  condition_token(T2,S),condition_token_path(Path2).


condition_property(S,in_path_has_patternlist(T,Path,PL))-->
  condition_token(T,S),condition_token_path(Path),[has,text],pattern_list(PL).
condition_property(S,in_path_contains_patternlist(T,Path,PL))-->
  condition_token(T,S),condition_token_path(Path),[contains,text],pattern_list(PL).
condition_property(S,in_path_contains_quantity(T,Path,Ls))-->
  condition_token(T,S),condition_token_path(Path),
  [contains,elements],quantity_type_list(Ls).
condition_property(S,in_path_has_quantity(T,Path,Ls))-->
  condition_token(T,S),condition_token_path(Path),
  [has,elements],quantity_type_list(Ls).
condition_property(S,in_path_is_in_docs(T,Path,Ls))-->
  condition_token(T,S),condition_token_path(Path),
  [is,in,one,of],general_list(Ls).
condition_property(S,elem_is_type_one_of(T,Ls))-->
```

```
  condition_token(T,S),[is,type,one,of],general_list(Ls).
condition_property(S,get_typed_position2(P,T,N))-->
  condition_token(T,S),[position,in],condition_token(P,S),[is,N].
condition_property(S,elems_has_fuzzy_similarity_by_content(T1,T2,Hedge,Mship))-->
          condition_token(T1,S),condition_token_path(P1),
          [is],fuzzy_hedge(Hedge),[like],condition_token(T2,S),condition_token_path(P2),
fuzzy_membership(Mship).

condition_property(S,Rule)-->derived_property(S,Rule).



element_value(D)-->[D],{chars(D);number(D)}.


condition_token_path_list([(T,Path)|Ts],S)-->
  ['('],condition_token(T,S),condition_token_path(Path),
  more_condition_path_tokens(Ts,S),[')'].

more_condition_path_tokens([],S)-->[].
more_condition_path_tokens([(T,Path)|Ts],S)-->
  [','],condition_token(T,S),condition_token_path(Path),
  more_condition_path_tokens(Ts,S).


condition_token_list([T|Ts],S)-->
  ['('],condition_token(T,S),more_condition_tokens(Ts,S),[')'].
more_condition_tokens([],S)-->[].
more_condition_tokens([T|Ts],S)-->
  [','],condition_token(T,S),more_condition_tokens(Ts,S).

condition_token(DataElement,S)-->[Variable],{member((Variable,DataElement,_),S)}.
condition_token(Doc,DataElement,S)-->[Variable],{member((Variable,DataElement,Doc),S)}.

pattern_list(Xs)-->['('],pattern_token(X),or_pattern_token(X,Xs),[')'].
pattern_token(Xs)-->pattern_elem(X),and_pattern_token(X,Xs).
or_pattern_token(X,X)-->[].
or_pattern_token(X1,or(X1,X2))-->[or],pattern_token(Xs),or_pattern_token(Xs,X2).
and_pattern_token(X,X)-->[].
and_pattern_token(X1,and(X1,X2))-->[and],pattern_token(Xs),and_pattern_token(Xs,X2).
pattern_token(Xs)-->[not],pattern_token(X),and_pattern_token(not(X),Xs).
pattern_elem(X)-->[X];pattern_list(X);one_of_list(X);near_list(X);fuzzy_sim_list(X).

%single_pattern_elem(X)-->[X];one_of_list(X);near_list(X).

one_of_list(one_of(X))-->[one,of],general_list(X).

near_list(near(X,Ls))-->[near,X,for],general_list(Ls).

fuzzy_sim_list( sim(Words,Mship,Hedge) )-->
    fuzzy_hedge(Hedge),[sim],general_list(Words),fuzzy_membership(Mship).

cond_path_id(_)-->[*].
cond_path_id(X)-->[X],{atom(X),not (X == *)}.

condition_token_path([])-->[].
condition_token_path([attr(X)])-->[/@,X].
condition_token_path([name])-->[/#,name].
condition_token_path([has(X)|Path])-->[/],cond_path_id(X),condition_token_path(Path).
condition_token_path([contains(X)|Path])-->
  [//],cond_path_id(X),condition_token_path(Path).

condition_token_path([has(X,Pos)|Path])-->
  [/],cond_path_id(X),position_pred(Pos),condition_token_path(Path).
condition_token_path([contains(X,Pos)|Path])-->
  [//],cond_path_id(X),position_pred(Pos),condition_token_path(Path).

quantity_type_list([X|Y])-->['('],quantity_token(X),more_quantity_tokens(Y),[')'].
more_quantity_tokens([])-->[].
more_quantity_tokens([L|Ls])-->[','],quantity_token(L),more_quantity_tokens(Ls).

quantity_token(eq(X,N))-->[X,N].
quantity_token(gt(X,N))-->[X,>,N].
quantity_token(lt(X,N))-->[X,<,N].
quantity_token(in(X,N1,N2))-->[X,in,N1,-,N2].
quantity_token(out(X,N1,N2))-->[X,out,N1,-,N2].


general_list([L|Ls])-->['(',L],more_general_tokens(Ls),[')'].

more_general_tokens([])-->[].
more_general_tokens([L|Ls])-->[',',L],more_general_tokens(Ls).
```

# Kyselytulkki

```
query:-
  write_title,prompt(Prompt,'query |: '),repeat,read_user_input(X),
  process_it(X),!,prompt(A,'|: ').

read_user_input(X):-etoks(TokenList),rip_it(TokenList,X).

process_it([quit]):-!,retractall(idnumber(_)).
process_it([cls]):-!,put(12),fail.
process_it([do,def]):-!,define_derived_property,write('Returning to the query
state.'),nl,fail.
process_it([save,def,File]):-!,save_derived_definition_to(File),write('Definitions
saved.'),nl,fail.
process_it([load,def,File]):-!,consult(File),nl,fail.
process_it([reset]):-!,dynamic(derived_property/4),write('All definitions
resetted.'),nl,fail.

process_it(X):-phrase(query(q_selection(Selection,Variables,Conditions)),X),
       findallvars(Variables,Conditions,Allvars),
       findall_tuples(Allvars,Selection,List),
       remove_duplicates(List,PlainRes),
       test_for_reconstruction(PlainRes,Restructured),
       write_result_elements(Restructured,0),fail.

process_it(X):-(\+ phrase(query(_),X)),nl,write('Syntax Error.'),nl,fail.

rip_it([],[]):-!.
rip_it([(8,',')|Xs],[',',|Ys]):-!,rip_it(Xs,Ys).
rip_it([(8,'(')|Xs],['(',|Ys]):-!,rip_it(Xs,Ys).
rip_it([(8,')')|Xs],[')'|Ys]):-!,rip_it(Xs,Ys).
rip_it([(8,'[')|Xs],['[',|Ys]):-!,rip_it(Xs,Ys).
rip_it([(8,']')|Xs],[']'|Ys]):-!,rip_it(Xs,Ys).
rip_it([(8,'|')|Xs],['|'|Ys]):-!,rip_it(Xs,Ys).
rip_it([(8,'{')|Xs],['{'|Ys]):-!,rip_it(Xs,Ys).
rip_it([(8,'}')|Xs],['}'|Ys]):-!,rip_it(Xs,Ys).
rip_it([(3,'/../')|Xs],['/..'|['/'|Ys]]):-!,rip_it(Xs,Ys).
rip_it([(3,'//../')|Xs],['//..'|['/'|Ys]]):-!,rip_it(Xs,Ys).
rip_it([(3,'/..//')|Xs],['/..'|['//'|Ys]]):-!,rip_it(Xs,Ys).
rip_it([(3,'//..//')|Xs],['//..'|['//'|Ys]]):-!,rip_it(Xs,Ys).
rip_it([(8,_)|Xs],Ys):-rip_it(Xs,Ys).
rip_it([(0,X)|Xs],[X|Ys]):-rip_it(Xs,Ys). %variable
rip_it([(1,X)|Xs],[X|Ys]):-rip_it(Xs,Ys). %integer
rip_it([(2,X)|Xs],[X|Ys]):-rip_it(Xs,Ys). %float
rip_it([(3,X)|Xs],[X|Ys]):-rip_it(Xs,Ys). %unq. atom
rip_it([(4,X)|Xs],[X|Ys]):-rip_it(Xs,Ys). %string
rip_it([(6,X)|Xs],[X|Ys]):-rip_it(Xs,Ys). %charstring
rip_it([(7,X)|Xs],[X|Ys]):-rip_it(Xs,Ys). %quoted atom


findallvars(Vars,Conds,Allvars):-
    create_find_list(Vars,Vfind),setof(Vars,(Vfind,Conds),Allvars).


findall_tuples(Allvars,selection(Res,Ss),List):-
   setof(X,get_selection_element(selection(Res,Ss),Allvars,X),List).

get_selection_element(selection(Res,Ss),Allvars,element(Res,[],X)):-
   member(OneVars,Allvars),build_simple_tuple(Ss,OneVars,X).
get_selection_element(m_sel(Res,Ss),Allvars,m_element(Res,[],X)):-
   member(OneVars,Allvars),build_simple_tuple(Ss,OneVars,X).
get_selection_element(plain_selection(Ss),Allvars,X):-
   member(OneVars,Allvars),build_simple_tuple(Ss,OneVars,X).


build_simple_tuple([],Vars,[]).
build_simple_tuple([(V,p(Path))|Ss],Vars,[E|Rs]):-
   member( (V,X,_) , Vars),selp_elem(X,Path,E),build_simple_tuple(Ss,Vars,Rs).
build_simple_tuple([pcdata(S)|Ss],Vars,[pcdata(S)|Rs]):-build_simple_tuple(Ss,Vars,Rs).
build_simple_tuple([empty(S)|Ss],Vars,[element(S,[],[])|Rs]):-
   build_simple_tuple(Ss,Vars,Rs).


build_simple_tuple([selection(Res,Sel)|Ss],Vars,[element(Res,[],X)|Rs]):-
   build_simple_tuple(Sel,Vars,X),build_simple_tuple(Ss,Vars,Rs).
build_simple_tuple([m_sel(Res,Sel)|Ss],Vars,[m_element(Res,[],X)|Rs]):-
   build_simple_tuple(Sel,Vars,X),build_simple_tuple(Ss,Vars,Rs).
```

```
test_for_reconstruction(Res,Res):-no_multituples(Res),!.
test_for_reconstruction(Ts,Rts):-
          reconstruct_tuples(Ts,Rets),
          test_for_reconstruction(Rets,Rts).


no_multituples([]).
no_multituples([element(_,_,T)|Ts]):-!,no_multi(T),no_multituples(Ts).
no_multituples([m_element(_,_,_)|Ts]):-!,fail.
no_multituples([_|Ts]):-no_multituples(Ts).
no_multi([]).
no_multi([element(_,_,E)|Ts]):-!,no_multi(E),no_multi(Ts).
no_multi([m_element(_,_,_)|Ts]):-!,fail.
no_multi([_|Ts]):-no_multi(Ts).


reconstruct_tuples([],[]).
reconstruct_tuples([Tuple|Ts],[Rtuple|Rts]):- reconstruct_tuple(Tuple,Rtuple,Ts,Nts),!,
                                                reconstruct_tuples(Nts,Rts).

reconstruct_tuple(Tuple,Rtuple,Ts,Nts):-
find_similar(Tuple,Ts,Sim,Nts),merge_similar(Tuple,Sim,Rtuple).



find_similar(Tuple,[],[],[]).
find_similar(Tuple,[T|Ts],[T|Sim],Nts):-
similar_tuple(Tuple,T),!,find_similar(Tuple,Ts,Sim,Nts).
find_similar(Tuple,[T|Ts],Sim,[T|Nts]):-find_similar(Tuple,Ts,Sim,Nts).

similar_tuple(element(R,A,E),element(R,A,E1)):-similar_list(E,E1).

similar_list([],[]).
similar_list([X|Xs],[X|Ys]):-similar_list(Xs,Ys).
similar_list([element(R,A,E)|Xs],[element(R,A,E1)|Ys]):-
similar_list(E,E1),similar_list(Xs,Ys).
similar_list([m_element(R,_,X)|Xs],[m_element(R,_,Y)|Ys]):-similar_list(Xs,Ys).


merge_similar(Tuple,[],Rtuple):-convert_m_element(Tuple,Rtuple).
merge_similar(Tuple,[S|Sims],Rtuple):-
  merge_tuple(Tuple,S,NTuple),merge_similar(NTuple,Sims,Rtuple).

merge_tuple(element(R,A,E),element(R,A,E1),element(R,A,M)):-
  merge_t_list(E,E1,M).
convert_m_element(element(R,A,E),element(R,A,E1)):-
  convert_m_list(E,E1).

merge_t_list([],X,X):-!.
merge_t_list(X,[],X):-!.
merge_t_list([element(R,A,E)|Xs],[element(R,A,E1)|Ys],[element(R,A,M)|Ms]):-
   !,merge_t_list(E,E1,M),merge_t_list(Xs,Ys,Ms).
merge_t_list([m_element(R,A1,E1)|Xs],[m_element(R,A2,E2)|Ys],[m_element(R,A,M)|Ms]):-
    similar_list(E1,E2),!,merge_tuple(element(R,A1,E1),
    element(R,A2,E2),element(R,A,M)),merge_t_list(Xs,Ys,Ms).
merge_t_list([m_element(R,A1,E1)|Xs],[m_element(R,A2,E2)|Ys],[m_element(R,Am,Em)|Ms]):-
    !,append(A1,A2,Am),append(E1,E2,Em),merge_t_list(Xs,Ys,Ms).
merge_t_list([X|Xs],[X|Ys],[X|Ms]):-!,merge_t_list(Xs,Ys,Ms).
merge_t_list([X|Xs],[Y|Ys],M):-!,merge_t_list(Xs,Ys,Ms),append([X,Y],Ms,M).

convert_m_list([],[]).
convert_m_list([element(Xn,Xa,Ex)|Xs],[element(Xn,Xa,Ey)|Ys]):-
   !,convert_m_list(Ex,Ey),convert_m_list(Xs,Ys).
convert_m_list([m_element(Xn,Xa,Ex)|Xs],[element(Xn,Xa,Ey)|Ys]):-
   !,convert_m_list(Ex,Ey),convert_m_list(Xs,Ys).
convert_m_list([conv(Els)|Xs],[New|Ys]):-
   !,handle_conversion(Els,New),convert_m_list(Xs,Ys).
convert_m_list([X|Xs],[X|Ys]):-convert_m_list(Xs,Ys).

handle_conversion([Elem],Elem):-!.
handle_conversion(Elems,element(N,As,Els)):-
   test_for_reconstruction(Elems,Res),do_final_merging(Res,N,As,Els).

do_final_merging([],_,[],[]).
do_final_merging([element(N,A,E)|Elms],N,Ats,Els):-
do_final_merging(Elms,_,As,Es),append(A,As,Ats),append(E,Es,Els).

create_find_list([],true).
create_find_list([(_,S,D)|Ss],(find_elem_type_from_docs(S,D),Sfs)):-
create_find_list(Ss,Sfs).

formalize_selection(element(X,A,E),element(X,A,Ef)):-formalize_elements(E,Ef).
```

```
formalize_elements([],[]).
formalize_elements([List|Es],Res):-
list(List),!,formalize_elements(List,Rl),formalize_elements(Es,Rs),append(Rl,Rs,Res).
formalize_elements([(An,Av)|Es],[element(An,[],[pcdata(As)])|Rs]):-
!,do_string(Av,As),formalize_elements(Es,Rs).
formalize_elements([element(Xn,Xa,Xe)|Es],[element(Xn,Xa,Xef)|Rs]):-
!,formalize_selection(element(Xn,Xa,Xe),element(Xn,Xa,Xef)),formalize_elements(Es,Rs).
formalize_elements([pcdata(S1)|[pcdata(S2)|Es]],Rs):-
!,cat([S1,S2],S3,_),formalize_elements([pcdata(S3)|Es],Rs).
formalize_elements([X|Es],[X|Rs]):-formalize_elements(Es,Rs).


write_result_elements([],1):-!,write('1 result total.'),nl.
write_result_elements([],N):-write(N), write(' results total.'),nl.
write_result_elements([T|Ts],N):-formalize_selection(T,Tform),write_as_xml(Tform),N1 is
N + 1,write_result_elements(Ts,N1).


remove_duplicates([],[]).
remove_duplicates([A|As],[A|Bs]):-removeall(A,As,Rs),remove_duplicates(Rs,Bs).

write_title:-nl,write('Welcome to Tagged Document Query Language parser'),nl.

define_derived_property:-
   prompt(Prompt,'define |: '),write('Please define a derived property:'),
   nl,etoks(TokenList),rip_it(TokenList,X),process_definition(X),prompt(_,Prompt).
```